# Kfz-Bussysteme – Physical und Data Link Layer 3

## 3.1 Controller Area Network CAN nach ISO 11898

CAN ist das derzeit am häufigsten eingesetzte Kfz-Bussystem sowohl für Low-Speed- als auch für High-Speed-Anwendungen.

### 3.1.1 Entwicklung von CAN

CAN (Controller Area Network) wurde von Bosch in der zweiten Hälfte der 80er Jahre entwickelt und seit 1991 als erstes Class C-Netz in Fahrzeugen eingesetzt [1, 2]. Kurz danach wurde die Spezifikation mit der Einführung der *29 bit Identifier* nochmals überarbeitet und stellt in der als CAN 2.0A und 2.0B von Bosch veröffentlichten Form bis heute die Basis aller existierenden CAN-Implementierungen dar. Mit dem ISO-Standard ISO 11898 und dem SAE-Standard SAE J2284 wurde das Protokoll für die Anwendung im PKW und mit dem SAE-Standard SAE J1939 für die Anwendung im NKW international standardisiert. Während CAN sich bei den europäischen PKW-Herstellern und bei praktisch allen NKW-Herstellern weltweit sehr schnell durchgesetzt hat, behielten amerikanische PKW-Hersteller auf dem heimischen Mark lange den Class B-Bus SAE J1850 bei. Allmählich stellen aber auch diese Hersteller vollständig auf CAN um. Bei Neufahrzeugen ist CAN ab 2008 die einzig zugelassene Schnittstelle für Diagnosetester bei abgasrelevanten Komponenten (OBD).

Die erste CAN-Implementierung entstand in Zusammenarbeit zwischen Bosch und Intel. Bosch hat aber bereits frühzeitig eine offene Lizenzpolitik betrieben, so dass mittlerweile praktisch jeder Mikrocontroller-Hersteller CAN-Module anbietet und sogar CAN-VHDL-Module für die Integration in ASICs und FPGAs zur Verfügung stehen. Bei der Lizenzierung stellt Bosch ein Referenzmodell und eine Testprozedur zur Verfügung, so dass sichergestellt ist, dass alle CAN-Controller kompatibel miteinander kommunizieren können.

W. Zimmermann und R. Schmidgall, *Bussysteme in der Fahrzeugtechnik*, ATZ/MTZ-Fachbuch, 57
DOI 10.1007/978-3-658-02419-2_3, © Springer Fachmedien Wiesbaden 2014

**Tab. 3.1** Wichtige CAN-Standards

| | |
|---|---|
| ISO 11898 – 1 | **Data Link Layer**, entspricht den Bosch Spezifikationen CAN 2.0A und CAN 2.0B |
| ISO 11898 – 2, 5, 6<br>ISO 11898 – 3 | **Physical Layer** für High Speed CAN<br>… und Low Speed CAN |
| ISO 11898 – 4 | Erweiterung des Data Link Layer für zeitgesteuerte Kommunikation (**Time Triggered CAN**) |

Durch die hohen Stückzahlen im Automobilbereich sind CAN-Controller viel preisgünstiger als die meisten ASICs für die in der Automatisierungstechnik verbreiteten Feldbusse wie Profibus. CAN wird daher auch in industriellen Anwendungen als Sensor-Aktor-Bus verwendet (*CAN in Automation*).

Zu den größten Schwächen der ursprünglichen CAN-Spezifikation von Bosch gehört, dass im Wesentlichen nur der *Data Link Layer* spezifiziert ist. Zur Bitrate wie zum *Physical Layer* gab es nur einige Hinweise, um unterschiedlichste Implementierungen zu ermöglichen. Zum *Application Layer* gab es bewusst gar keine Vorgaben. Wie nicht anders zu erwarten, führten diese Freiheitsgrade zu unterschiedlichen Lösungen. Dabei resultieren die Unterschiede beim *Physical Layer* vor allem aus dem notwendigen Kompromiss zwischen gewünschter Buslänge und möglicher Bitrate und sind relativ überschaubar, während die Lösungen beim *Application Layer* weitgehend inkompatibel zueinander sind. Der *Physical Layer* und der *Data Link Layer* einschließlich der für emissionsrelevante Komponenten wesentlichen Einschränkungen werden in den folgenden beiden Abschnitten beschrieben, der *Transport Layer* in Kap. 4 und der *Application Layer* in Kap. 5. Die gesamte Beschreibung soll einen Überblick geben, zu konkreten Details wird auf die Originaldokumente verwiesen (Tab. 3.1).

### 3.1.2 Bus-Topologie und Physical Layer

CAN ist ein bitstrom-orientierter Linien-Bus, der für eine maximale Bitrate von 1 Mbit/s definiert ist. CAN verwendet einen CSMA/CR-Buszugriff sowie eine Fehlererkennung, die die Reaktion aller Steuergeräte innerhalb einer Bitzeit erfordert. Dadurch muss die Buslänge umso kleiner sein, je höher die Bitrate ist. Die einzuhaltende Bedingung ist

$$\text{Buslänge} \leq 40 \ldots 50\,\text{m} \cdot \frac{1\,\text{Mbit/s}}{\text{Bitrate}}. \tag{3.1}$$

Die Formel ist eine Faustformel, da bei hohen Bitraten die Verzögerungszeiten der Bustranceiver oder der bei sehr großen Buslängen in der Automatisierungstechnik gegebenenfalls eingesetzten Repeater die zulässige Buslänge bzw. Bitrate reduzieren. Alle Steuergeräte des Busses müssen mit derselben Bitrate arbeiten.

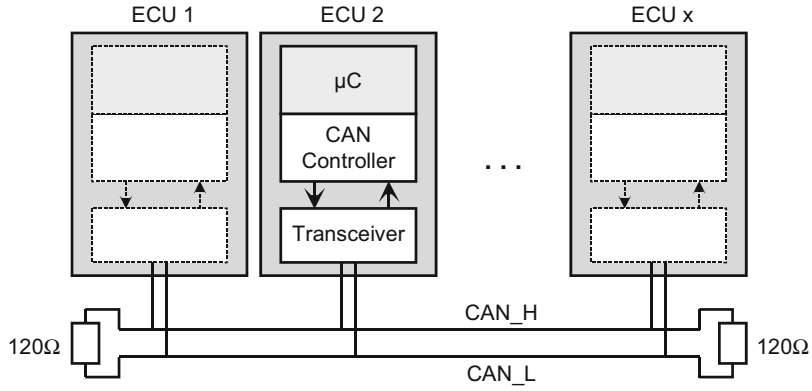


**Abb. 3.1** High-Speed-CAN-Bus

Der Anhang ISO 11898-2 zum CAN-ISO-Standard definiert für Applikationen mit Bitraten ≥ 250 kbit/s (*High Speed CAN*, d. h. Class C) die Verwendung einer nach Möglichkeit verdrillten Zwei-Draht-Leitung als echter Linien-Bus mit Stichleitungen von max. 30 cm zu den einzelnen Steuergeräten. Der Bus muss an beiden Enden mit dem Wellenwiderstand der Zwei-Draht-Leitung, typischerweise ca. 120 Ω, abgeschlossen werden (Abb. 3.1). Der Signalhub des Differenzspannungssignals liegt bei ca. 2 V (Abb. 3.2). Bei Unterbrechung oder Kurzschluss einer Ader der Zwei-Draht-Leitung fällt der Bus aus. In PKWs werden *High-Speed-CAN*-Busse im Antriebsstrang-Bereich mit Bitraten von typischerweise 500 kbit/s, bei Nutzfahrzeugen auch 250 kbit/s eingesetzt (Class C). Diese Bitraten sind auch in den SAE-Normen für CAN in PKW (SAE J2284) bzw. LKW (SAE J1939) standardisiert. Die neueren Zusatznormen ISO 11898-5 und ISO 11898-6 definieren verschiedene Maßnahmen zur Energieeinsparung.

Im Anhang ISO 11898-3 wird für Bitraten ≤ 125 kbit/s (*Low Speed CAN*, d. h. Class B), z. B. für Anwendungen in der Karosserieelektronik, ebenfalls eine Zwei-Draht-Leitungsverbindung spezifiziert. Aufgrund der niedrigeren Bitrate darf der Bus entsprechend länger sein. Die Busabschlusswiderstände und die Begrenzung auf kurze Stichleitungen entfallen. Der Signalhub des Differenzsignals ist deutlich größer als beim *High Speed CAN* (Abb. 3.2). Der Bus bleibt auch bei Kabelbruch oder Kurzschluss einer Ader noch funktionsfähig. *Low*

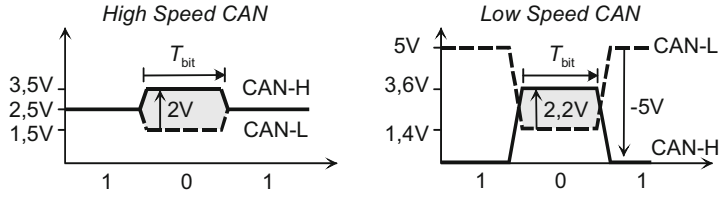


**Abb. 3.2** Signalpegel bei High- und bei Low-Speed-CAN nach ISO 11898-2 bzw. 3

*Speed CAN* wird in europäischen Kfz in der Regel mit 100 bis 125 kbit/s für die Karosserieelektronik eingesetzt.

In SAE J2411 ist auch eine Ein-Draht-Ausführung mit Bitraten von 33 kbit/s (Einsatz bei GM) und 83 kbit/s (Einsatz bei Chrysler) und einem Signalhub von 5 V für die Kfz-Komfortelektronik spezifiziert (*Single Wire CAN*).

Für die Verbindung von LKW-Zugmaschinen und Anhängern (*Truck and Trailer*) wird in ISO 11992 eine Punkt-zu-Punkt-Zwei-Draht-Ausführung mit 125 kbit/s und einem Signalpegel in der Größenordnung der Batteriespannung definiert.

Für Anwendungen in Nutzfahrzeugen spezifiziert SAE J1939/11 eine feste Bitrate von 250 kbit/s mit einer Busankopplung, die weitgehend der High Speed CAN Spezifikation von ISO 11898-2 entspricht, allerdings wird eine Abschirmung der Zwei-Draht-Leitung, eine maximale Buslänge von 40 m und eine Höchstzahl von max. 30 Steuergeräten gefordert. In SAE J1939/12 wird eine Variante ohne abgeschirmtes Kabel definiert. Der in SAE J1939/21 spezifizierte Data Link Layer entspricht dem nachfolgend beschriebenen CAN 2.0B.

Der CAN-Anschluss an Kfz-Steuergeräte erfolgt üblicherweise über den normalen Steuergerätestecker. Für die Automatisierungstechnik definiert CAN-in-Automation (CiA) in der Empfehlung CiA DS 102 für Bitraten zwischen 10 kbit/s … 1 Mbit/s einen Physical Layer ähnlich ISO 11898-2. DeviceNet verwendet 125 … 500 kbit/s. In der Automatisierungstechnik sind CAN-Anschlüsse über 9 polige Sub-Miniatur-D-Stecker und verschiedene andere Steckertypen im Einsatz.

Für alle CAN-Ausführungsformen gibt es auf dem Markt geeignete Transceiver-Bausteine. Untereinander sind die verschiedenen Varianten des Physical Layer nicht kompatibel, da die Signalpegel unterschiedlich sind (Abb. 3.2). In allen Fällen werden diejenigen Signalpegel hochohmig (*rezessiv*) erzeugt, die eine logische „1“ definieren. Signalpegel, die die logische „0“ übertragen, sind dagegen niederohmig (*dominant*).

Im Gegensatz zu einfachen UARTs kann und muss der Abtastzeitpunkt des Bussignals (*Bit Timing*) bei den meisten CAN-Controllern vom Anwender eingestellt werden (Abb. 3.3). Die Taktperiode, aus der der Bittakt $T_{\mathrm{bit}}$ abgeleitet wird, wird als *Quantum* $T_{\mathrm{Q}}$ bezeichnet. Im ersten *Quantum*, dem *Synchronization Segment* $T_{\mathrm{SyncSeg}}$, gibt der Sender die Signalflanke aus. Die Signallaufzeiten durch die beiden Transceiver im Sender und im Empfänger und auf der Busleitung werden durch das *Propagation Segment* $T_{\mathrm{PropSeg}}$ berücksichtigt. Wegen der *bitweisen Arbitrierung*, die im folgenden Abschnitt beschrieben wird, muss $T_{\mathrm{SyncSeg}} + T_{\mathrm{ProgSeg}}$ mindestens doppelt so lang sein wie die maximale Signalverzögerung. Die Abtastung des Bits erfolgt dann ungefähr in der Mitte der verbleibenden Zeit $T_{\mathrm{PhaseSeg1}} + T_{\mathrm{PhaseSeg2}}$ bis zum Bitende. Die Dauer der beiden *Phase Segment* Abschnitte kann vom CAN-Controller dynamisch verlängert bzw. verkürzt werden, um den Bittakt des Empfängers automatisch auf den Bittakt des Senders zu synchronisieren. Die Anpassung erfolgt in Schritten von $T_{\mathrm{Q}}$, maximal um die sogenannte *Synchronisation Jump Width* $T_{\mathrm{SJW}} = 1 \ldots \min(4T_{\mathrm{Q}}, T_{\mathrm{PhaseSeg1}})$. Da das *Bit Timing* insbesondere für Softwareentwickler etwas unübersichtlich ist, enthält z. B. die Norm ISO 15765-4 für die Diagnose abgasrelevanter Systeme über CAN verschiedene Empfehlungen. Bei einer Bitrate von

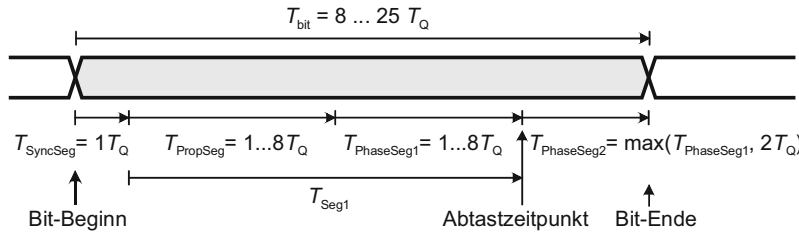


**Abb. 3.3** Aufteilung eines CAN-Bits in Zeitabschnitte

500 kbit/s mit einer Toleranz von ± 0,15 %, wie sie für Diagnosetester gefordert wird, und einem *Quantum* $T_Q = 125$ ns ergibt sich eine Bitdauer von $T_{bit} = 16\ T_Q$. Dabei wird empfohlen, $T_{Seg1} = T_{PropSeg} + T_{PhaseSeg1} = 12\ T_Q$ und $T_{Seg2} = T_{PhaseSeg2} = 3T_Q$ mit $T_{SJW} = 3\ T_Q$ einzustellen.

### 3.1.3 CAN Data Link Layer

Beim bitstrom-orientierten CAN-Protokoll erfolgt die gesamte Botschaftsübertragung selbstständig durch den Kommunikationsbaustein (CAN-Controller). Die Einzelheiten der Bitübertragung sind daher lediglich für den Entwickler eines CAN-Controllers von Bedeutung, während der CAN-Anwender lediglich eine grobe Vorstellung vom Botschaftsaufbau und Übertragungsablauf haben muss (Abb. 3.4).

CAN ist ein Broadcast-System, bei dem jeder Sender seine Botschaften ohne Ziel- und Quelladresse absendet, sondern jede Botschaft durch eine eindeutige Kennung, den *Message Identifier*, markiert. Ein Verbindungsaufbau ist nicht notwendig. Jedes Steuergerät am Bus empfängt die Botschaft und entscheidet anhand des *Message Identifiers*, ob es die Botschaft weiterverarbeitet oder ignoriert. Die Länge des *Message Identifiers* war ursprünglich 11 bit (CAN 2.0A), in der zweiten CAN-Generation wurden aber zusätzlich auch aufwärts kompatible 29 bit-Identifier (CAN 2.0B) definiert. Daneben werden im Identifier-Feld noch 1 bzw. 3 Steuerbits übertragen.

Beim Buszugriff wird das CSMA/CR-Verfahren verwendet. Jedes Steuergerät kann senden, sobald der Bus für mindestens 3 Bitzeiten frei ist. Der *Message Identifier* kenn-

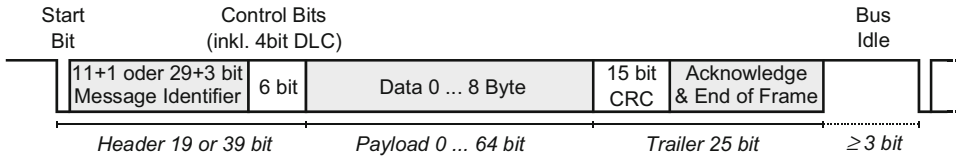


**Abb. 3.4** Botschaftsformat (Längenangaben ohne Bit-Stuffing, siehe unten)

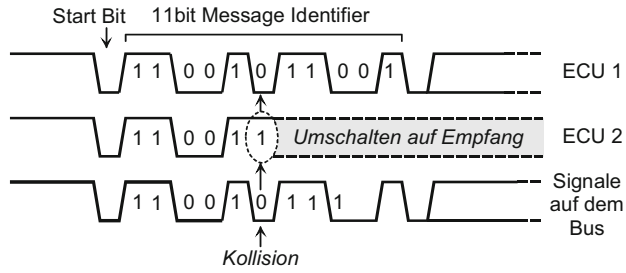


**Abb. 3.5** Kollision von CAN-Botschaften

zeichnet neben dem Inhalt einer Botschaft auch deren Priorität (niedrigere Zahl bedeutet höhere Priorität). Kommt es dabei zu einer Kollision, so „gewinnt" die Botschaft mit der höheren Priorität, d. h. der Sender, dessen *Message Identifier* den kleineren Wert hat. Im Beispiel nach Abb. 3.5 beginnen das Steuergerät ECU 1 mit dem *Message Identifier* 110 0101 $1001_B = 659_H$ und das Steuergerät ECU 2 mit dem *Message Identifier* 110 0111 $0000_B = 670_H$ gleichzeitig mit dem Senden. Beim sechsten Bit des *Message Identifiers* kommt es zu einer *Kollision*, weil ECU 2 eine 1 senden will, während ECU 1 eine 0 sendet. Da das 0-Signal im Vergleich zum 1-Signal wesentlich niederohmiger ist, wie bereits im vorigen Abschnitt erläutert wurde, dominiert auf den Busleitungen das 0-Signal. Weil die Steuergeräte beim Senden ständig überprüfen, ob der Signalpegel auf dem Bus dem jeweils gesendeten Bit tatsächlich entspricht, erkennt ECU 2 sofort, dass es zu einer Kollision gekommen ist, stellt das Senden ein und schaltet auf Empfang um. Die Kollisionserkennung und Auflösung wird als *bitweise Arbitrierung* bezeichnet. Die Übertragung der Botschaft des Steuergerätes ECU1 wird unverzögert fortgesetzt, während ECU 2 frühestens dann erneut mit dem Senden beginnt, wenn der Bus nach Ende dieser Botschaft wieder frei ist.

Eine Botschaft kann 0 bis 8 Nutzdatenbytes (*Payload*) übertragen (Abb. 3.4). Die Anzahl steht im *Data Length Code* (DLC) Feld innerhalb der Steuerbits (*Control Bits*). Zur Fehlererkennung wird eine 15 bit lange Prüfsumme (*Cyclic Redundancy Check*) mitgesendet.

Die Empfänger synchronisieren ihren Bittaktgenerator über das Startbit mit dem Sender und werden durch zusätzlich eingefügte, so genannte *Stuff-Bits* nachsynchronisiert. Die Anzahl der *Stuff-Bits* hängt vom den übertragenen Daten ab. Im theoretisch ungünstigsten Fall wird nach jedem 5. Bit ein *Stuff-Bit* so eingefügt, dass niemals mehr als fünf aufeinanderfolgende Bits denselben Wert aufweisen. Da die *Stuff-Bits* einerseits selbst wieder Teil des nächsten *Stuffing*-Blocks sind, andererseits aber das Botschaftsende nach der CRC-Prüfsumme nicht mehr dem *Stuffing* unterliegt, verlängert sich die Botschaft durch das *Stuffing* im ungünstigsten Fall um knapp 25 % [5, 6]. Auf der Empfangsseite werden die Stuff-Bits automatisch wieder entfernt.

Die CAN-Controller der am Bus angeschlossenen Steuergeräte überprüfen das Botschaftsformat und die Prüfsumme und senden innerhalb des *Acknowledge* und *End of Frame* Felds eine positive Empfangsbestätigung oder Fehlermeldung (*Error Frame*). Bei einer Fehlermeldung ignorieren alle Empfänger die empfangenen Daten. Dadurch ist sicher-

gestellt, dass die Daten im gesamten Netzwerk konsistent bleiben. Der Sender, der die Fehlermeldung erhält, startet automatisch einen neuen Sendeversuch.

Eine spezielle Botschaftsform ist der *Remote Frame*. Mit diesem Frame, der einen üblichen *Message Identifier*, aber keine Nutzdaten enthält und bei dem ein Bit am Ende des *Identifier* Felds gesetzt wird, fordert ein Steuergerät eine Botschaft mit den zu diesem *Message Identifier* gehörenden Daten von einem anderen Steuergerät an.

### 3.1.4 Fehlerbehandlung

Die verschiedenen Fehlererkennungsverfahren sorgen für eine hohe Übertragungszuverlässigkeit. Nach verschiedenen Untersuchungen liegt die Restfehlerwahrscheinlichkeit deutlich unter $10^{-11}$. Da Fehler unmittelbar nach Erkennung, spätestens am Ende einer Botschaft signalisiert werden, erfolgt sehr schnell eine automatische Übertragungswiederholung.

Jeder CAN-Controller enthält Fehlerzähler, mit deren Hilfe er eigene Sende- und Empfangsfehler protokolliert und Fehlermeldungen sendet (*Error active*). Dabei unterscheidet er auch, ob er die Fehler selbst erkannt hat oder ob sie auch von anderen CAN-Controllern festgestellt wurden. Falls der CAN-Controller dadurch erkennt, dass er selbst fehlerhaft arbeitet, stellt er zunächst eigene Sendeversuche von Fehlermeldungen ein (*Error passive*) und schaltet sich bei anhaltenden Problemen vollständig ab (*Bus off*). Verschwindet das Problem, z. B. weil es durch eine vorübergehende EMV-Störung bewirkt wurde, so aktiviert sich der CAN-Controller wieder.

### 3.1.5 Einsatz von CAN – Höhere Protokolle

Mit der relativ kurzen Botschaftslänge, dem prioritätsgesteuerten Buszugriff und der hohen Fehlersicherheit wurde CAN bewusst für den Austausch von Mess-, Regel- und Steuerdaten im Echtzeitbetrieb konzipiert und war ursprünglich hauptsächlich als Ersatz für Punkt-zu-Punkt-Verbindungen mit analogen und digitalen Signalen im Kfz vorgesehen. Da für solche Anwendungen höchste Flexibilität und geringstmöglicher Overhead gefordert sind, machten die ursprünglichen CAN-Spezifikationen keinerlei Vorgaben für die Vergabe der Message Identifier oder die Bedeutung, Formatierung und Normierung der übertragenen Daten.

Im PKW-Bereich legte und legt praktisch jeder europäische PKW-Hersteller diese Punkte individuell fest. Als Hilfsmittel dafür diente zunächst die so genannte *CAN- oder Kommunikations-Matrix*, in der in einer Tabellenstruktur dargestellt wird,

- welche Steuergeräte welche Botschaften unter welchen Bedingungen und mit welcher Zykluszeit senden,
- welche Steuergeräte diese Botschaften weiterverarbeiten,

- welche Daten (*Signale*) in diesen Botschaften enthalten sind und wie diese Daten normiert sind, d. h. welche Umrechnungsbeziehung zwischen den hexadezimalen Werten und den realen physikalischen oder logischen Größen bestehen,
- mit welcher Priorität, d. h. mit welchen Message Identifiern, die Botschaften gesendet werden.

Die so festgelegten Parameter müssen zumindest innerhalb eines Fahrzeugmodells für alle Steuergeräte einheitlich sein und von den verschiedenen Steuergeräte-Zulieferern gleich implementiert werden. Dafür stellen die Hersteller von CAN-Entwicklungs- und Testumgebungen heute datenbankgestützte Werkzeuge für die Erstellung und Pflege der *CAN-Matrix* (CAN-Datenbank CANdb, FIBEX-Beschreibungen) bereit.

Heute müssen bereits bei einem einzelnen Kfz-Steuergerät während der Entwicklungs- und Applikationsphase mehrere Tausend Datenwerte modellspezifisch festgelegt werden. Daher werden zunehmend automatisierte Applikationswerkzeuge eingesetzt. Die Kfz-Industrie bemüht sich im ASAM-Konsortium, die Daten-Darstellung (Formatierung, Normierung) zumindest auf der Ebene der Datenaustauschformate dieser Applikationswerkzeuge zu standardisieren.

Auf der NKW-Seite wurde die Problematik wesentlich früher erkannt. Basierend auf den Erfahrungen mit dem in USA schon Anfang der 90er Jahre verbreiteten zeichenbasierten Class A-Protokoll J1587/J1708 wurden die oben genannten Punkte für CAN Mitte der 90er Jahre in der Normenfamilie SAE J1939 einheitlich spezifiziert (siehe Abschn. 4.5)

Aufgrund der vorhandenen Infrastruktur in Werkstätten und Fertigungen, die bevorzugt auf der K-Line-Schnittstelle basierte, und des bis Ende der 90er Jahre noch nicht durchgängigen Einsatzes von CAN in allen Kfz-Steuergeräten, galt CAN für die Anwendungsgebiete Diagnoseschnittstelle und End-of-Line-/Flash-Programmierung zunächst als uninteressant. Die CAN-Spezifikationen trafen daher auch keine Vorkehrungen für den Transport von zusammengehörenden Datenblöcken von mehr als 8 Byte. Mittlerweile wird CAN auch auf diesem Gebiet verwendet. Die notwendigen Voraussetzungen dazu schafft die Normenfamilie ISO 15765, die beschreibt, wie das KWP 2000 bzw. UDS-Protokoll über CAN realisiert werden kann und in ISO 15765-2 auch ein Transportprotokoll (TP) definiert, das die Aufteilung von Datenblöcken von bis zu 4 K-Byte in einzelne CAN-Botschaften ermöglicht (siehe Abschn. 4.1).

### 3.1.6 Schnittstelle zwischen Protokoll-Software und CAN-Controller

Die gesamte Botschaftsübertragung einschließlich Fehlerüberprüfung erfolgt selbstständig durch den Kommunikationsbaustein (*CAN-Controller*), der heute in der Regel als On-Chip-Modul der eingesetzten Mikrocontroller implementiert ist, während die physikalische Ankopplung an die Busleitungen durch einen separaten CAN-Transceiver-Baustein erfolgt (Abb. 3.1). Praktisch alle neueren CAN-Controller können so konfiguriert werden,

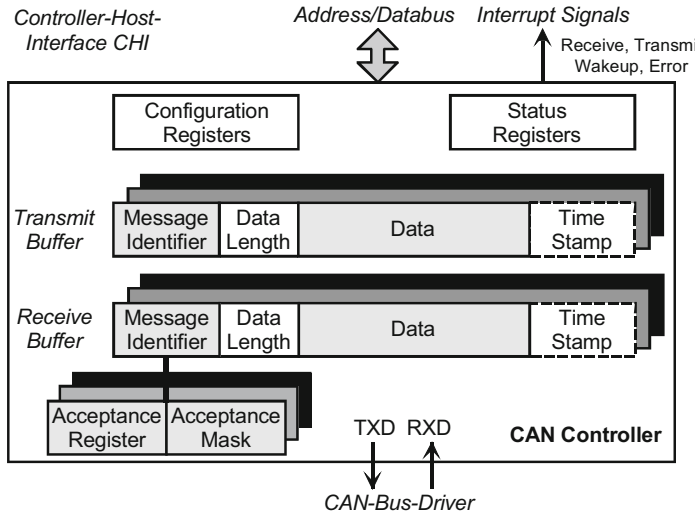


**Abb. 3.6** Typischer CAN-Kommunikationscontroller

dass sie sowohl mit 11 bit als auch mit 29 bit Message Identifiern arbeiten können, d. h. CAN 2.0A- und CAN 2.0B-fähig sind.

Aus Softwaresicht stellt der CAN-Controller eine Reihe von Steuerregistern sowie Speicherbereiche für mehrere Botschaften bereit (Abb. 3.6). Für die Übertragung einer Botschaft schreibt die Software den Message Identifier, die Anzahl der Datenbytes und die Datenbytes selbst in den Botschaftsspeicher, das Absenden erfolgt dann selbstständig durch den Controller. Beim Empfang liest die Software die Werte aus dem Botschaftsspeicher aus. Mit Hilfe von Statusbits und Interrupts synchronisieren CAN-Controller und Mikroprozessor den Zugriff auf den Botschaftsspeicher (Handshake) und tauschen Fehlermeldungen aus.

Ursprünglich boten die Halbleiterhersteller zwei verschiedene CAN-Controllertypen an, die auf der Busseite zueinander kompatibel sind und daher im selben CAN-Netz problemlos miteinander kommunizieren können, sich aber im Bereich der Akzeptanzfilterung (siehe unten) und damit im Schaltungs- und Softwareaufwand unterschieden:

- Reine *Basic CAN*-Controller verfügen typischerweise über je einen Speicherbereich für eine zu sendende und eine zu empfangende Botschaft. Die Akzeptanzfilterung, d. h. das Auswerten des Message Identifiers einer empfangenen Botschaft und die Entscheidung, ob die Botschaft überhaupt vom entsprechenden Steuergerät verarbeitet werden muss, erfolgt durch die CAN-Software des Steuergerätes. Um die Softwarebelastung beim Empfang etwas zu reduzieren, verfügen Basic-CAN-Controller aber praktisch immer über eine Möglichkeit, Botschaften mit bestimmten Message Identifiern zu ignorieren. In der Regel wird dies durch eine Bitmaske für den Message Identifier realisiert.

Die Speicherbereiche für die Sende- und die Empfangsbotschaft sind in der Regel doppelt gepuffert (Double Buffer), so dass bereits eine neue Botschaft in den jeweiligen Botschaftsspeicher geschrieben bzw. empfangen werden kann, bevor die vorige Botschaft vollständig gesendet bzw. durch die Software weiterverarbeitet ist. Abhängig von der Implementierungsstrategie des Halbleiterherstellers wird eine vorige Botschaft überschrieben oder die neueste Botschaft ignoriert, wenn der Botschaftsspeicher nicht frei ist. Seltener findet man FIFO-(First In First Out)-Strukturen für den Sende- und Empfangsspeicher, weil das Prioritätskonzept von CAN es eigentlich notwendig macht, dass eine später empfangene oder bereitgestellte Botschaft früher verarbeitet oder versendet wird als eine frühere Botschaft, wenn die spätere Botschaft eine höhere Priorität hat. Dazu wäre ein dynamisches Umorganisieren eines FIFO-artig betriebenen Botschaftsspeichers notwendig. Die Priorisierung der Botschaften ist hinsichtlich der Echtzeiteigenschaften nur dann wirklich konsistent, wenn sie nicht nur beim Buszugriff, sondern auch bei der Botschaftsverarbeitung in der CAN-Software konsequent berücksichtigt wird.

- *Full CAN*-Controller haben typischerweise 8 bis 16 Botschaftsspeicher, wobei jeder Botschaftsspeicher entweder für das Senden oder Empfangen einer bestimmten Botschaft konfiguriert werden kann. Die Akzeptanzfilterung findet automatisch, d. h. in der CAN-Controller-Hardware statt, wobei wiederum für jeden Botschaftsspeicher konfiguriert werden kann, welche Message Identifier akzeptiert werden. Neu ankommende Botschaften überschreiben dabei in der Regel die vorige Botschaft im entsprechenden Botschaftsspeicher. Durch die hardwaremäßige Akzeptanzfilterung ist die Softwarebelastung beim Empfang deutlich geringer als bei *Basic CAN*. Durch die verschiedenen Botschaftsspeicher ist mehr Zeit für die Weiterverarbeitung empfangener Botschaften vorhanden und die Bereitstellung der zu sendenden Botschaften kann gruppenweise und teilweise unabhängig von der Prioritätenreihenfolge erfolgen, da der CAN-Controller unter den zu sendenden Botschaften stets die höchstpriore als erste auswählt.

Heutige CAN-Controller (Abb. 3.6) sind meist Mischformen. In der Regel verfügen sie über Speicher für eine Hand voll verschiedene Botschaften mit Hardware-Akzeptanzfilterung (*Full CAN*) sowie Speicher für mindestens eine Sende- und eine Empfangsbotschaft mit Software-Akzeptanzfilterung (*Basic CAN*). Bevor Botschaften versendet und empfangen werden, wird der Kommunikationscontroller durch den steuernden Mikrocontroller (*Host*) initialisiert. Dabei werden Bitrate und Bittiming nach Abb. 3.3 sowie die Interrupts eingestellt, die Akzeptanzfilter gesetzt und der Betriebsmodus ausgewählt. Neben dem Normalbetrieb können CAN-Controller häufig auch so betrieben werden, dass sie Botschaften nur empfangen (*Listen* oder *Silent Mode*), aber nicht am normalen Busbetrieb teilnehmen, d. h. auch keine Empfangsbestätigung (*Acknowledge*) und keine Fehlermeldungen senden. Dieser Modus wird zur Analyse des Datenverkehrs eingesetzt, ohne den Bus zu beeinflussen. Zu Testzwecken in der Entwicklungsphase wird der *Loopback Mode* eingesetzt, bei dem der CAN Controller die Botschaften, die er sendet, nur selbst empfängt, den Bustreiber dagegen komplett abschaltet. Außerdem kann der CAN Controller

**Tab. 3.2** Beispiel für die Einstellung eines Akzeptanzfilters für 11 bit Identifier

| *Bit* | *10* | *9* | *8* | *7* | *6* | *5* | *4* | *3* | *2* | *1* | *0* |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Akzeptanzregister AR | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| Akzeptanzmaske AM | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| Effektives Akzeptanzfilter AF | 0 | 1 | 1 | 0 | 1 | 1 | 1 | X | X | X | X |

37Xh = 370 … 37Fh

in einen stromsparenden Modus (*Sleep Mode*) geschaltet werden, aus dem er durch den Mikrocontroller oder durch den Empfang einer beliebigen CAN-Botschaft wieder aufgeweckt (*Wakeup*) wird.

Nach dem Verlassen des Konfigurationsmodus kann der Mikrocontroller Botschaften bestehend aus dem Message Identifier, den Datenbytes und deren Anzahl in einen der Sendespeicher schreiben und durch Setzen des zugehörigen Statusbits zum Senden freigeben. Stehen in verschiedenen Sendespeicher mehrere Botschaften zum Senden an, versendet der Kommunikationscontroller je nach Ausführung die Botschaften in der Reihenfolge ihrer Priorität oder der Sendeanforderung. Ob die Botschaft erfolgreich versendet wurde, kann der Mikrocontroller durch Lesen eines Statusbits abfragen (*Polling*) oder sich automatisch (*Transmit Interrupt*) informieren lassen. Optional kann er zusätzlich den Zeitpunkt, zu dem einer der Busteilnehmer den Empfang durch Setzen des *Acknowledge Bit* im Botschafts-Trailer bestätigt hat, aus dem *Time Stamp* Feld des Sendespeichers auslesen. Analog wird der Empfangszeitpunkt einer Botschaft im *Time Stamp* Feld des Empfangsspeichers festgehalten. Welche Botschaften empfangen werden, wird durch die Akzeptanzfilter vorgegeben (Tab. 3.2). Jedes Akzeptanzfilter besteht in der Regel aus einem Register AR, in das der zu empfangene Message Identifier eingetragen wird, und einer Maske AM, die angibt, welche Bits des Message Identifiers überhaupt geprüft werden sollen. Auf diese Weise können Don't Care Bits festgelegt und damit ganze Identifier-Bereiche ausgewählt werden. In dem Beispiel nach Tab. 3.2 wird das Akzeptanzfilter auf 37Xh gesetzt und damit ein Bereich von 370h … 37Fh aktiviert. Der Mikrocontroller erfährt vom Empfang einer Botschaft entweder durch Abfragen des Statusregisters oder durch den *Receive Interrupt*. In ähnlicher Form kann sich der Mikrocontroller informieren oder informieren lassen, wenn eine Botschaft verloren geht, weil alle Empfangsspeicher voll waren, oder wenn es Fehler auf dem CAN-Bus gab.

### 3.1.7 Zeitverhalten von CAN-Systemen, Wahl der Botschaftspriorität

Die Länge einer CAN Botschaft einschließlich der 3 bit langen *Bus Idle* Phase zwischen zwei aufeinanderfolgenden Botschaften ist

$$T_{\text{Frame}} = n_{\text{Frame}} \cdot T_{\text{bit}} = \left(n_{\text{Header}} + n_{\text{Trailer}} + n_{\text{Idle}} + n_{\text{Data}} + n_{\text{Stuff}}\right) \cdot T_{\text{bit}}. \qquad (3.2)$$

**Tab. 3.3** Botschaftslänge und Datenrate für $f_{\text{bit}} = 1 / T_{\text{bit}} = 500$ kbit/s

| | | Länge ohne Stuffing | | Länge mit Stuffing | | |
|---|---|---|---|---|---|---|
| CAN ID | $n_{\text{Data}}$ | $n_{\text{Frame, min}}$ | $T_{\text{Frame, min}}$ | $n_{\text{Frame, max}}$ | $T_{\text{Frame, max}}$ | $f_{\text{Data}} = n_{\text{Data}} / T_{\text{Frame}}$ |
| 11 bit | 1 byte | 55 bit | 110 µs | 65 bit | 130 µs | 7,5 KB/s |
| | 8 byte | 111 bit | 222 µs | 135 bit | 270 µs | 28,9 KB/s |
| 29 bit | 1 byte | 75 bit | 150 µs | 90 bit | 180 µs | 5,4 KB/s |
| | 8 byte | 131 bit | 262 µs | 160 bit | 320 µs | 24,4 KB/s |

Dabei ist $n_{\text{Header}} + n_{\text{Trailer}} + n_{\text{Idle}} = 47$ bit (67 bit), wenn mit 11 bit *Message Identifiern* gearbeitet wird. Die Werte für 29 bit Identifier sind in (…) angegeben. Die Länge des Nutzdatenbereichs $n_{\text{Data}} = 0, \ldots, 64$ bit ist in Stufen von 8 bit variabel.

Die Anzahl der Stuff-Bits hängt vom Botschaftsinhalt ab und beträgt nach [5, 6]

$$n_{\text{Stuff}} = 0 \ldots \left\lfloor \frac{n_{\text{Header}} + n_{\text{Trailer}} + n_{\text{Idle}} + n_{\text{Data}} - 14\,\text{bit}}{4} \right\rfloor, \tag{3.3}$$

wobei $\lfloor ... \rfloor$ bedeutet, dass der Bruch auf den nächsten ganzzahligen Wert abgerundet werden muss. Als Faustformel ergibt sich

$$T_{\text{Frame}} < 1,25 \cdot (47\,\text{bit}(67\,\text{bit}) + n_{\text{Data}}) \cdot T_{\text{bit}}. \tag{3.4}$$

Tabelle 3.3 gibt einen Überblick über die minimale und maximale Länge und die zugehörige Datenrate $f_{\text{Data}} = n_{\text{Data}} / T_{\text{Frame}}$.

CAN arbeitet ereignisgesteuert. Im Idealfall wird die Botschaft sofort gesendet, sobald sie in den Botschaftspuffer des Kommunikationscontrollers kopiert und das entsprechende Sendebit gesetzt wurde. Vernachlässigt man die Signallaufzeiten der Transceiver und der Leitung gegenüber der Botschaftsdauer, so ist die Zeit zwischen Bereitstellen der Botschaft auf der Sendeseite und deren Verfügbarkeit auf der Empfangsseite, d. h. die Buslatenz, im günstigsten Fall (vergleiche Abb. 2.17):

$$T_{\text{Latenz,min}} = T_{\text{Frame}}. \tag{3.5}$$

Wenn der Bus jedoch gerade belegt ist oder wenn eine andere Botschaft mit höherer Priorität zum Senden ansteht, muss die Botschaft warten, bis der Bus frei wird. Da das CAN Protokoll keine inhärente Obergrenze für diese Wartezeit garantiert, gilt der CAN-Bus als nicht-deterministisch (nicht *hart* echtzeitfähig) und daher im strengen Sinn nicht geeignet für Anwendungen wie Fahrwerksregelungen mit hohen Anforderungen an die zeitliche Präzision. Theoretische Arbeiten von Tindell, Burns und anderen [4–7] und langjährige praktische Erfahrungen haben gezeigt, dass sich unter bestimmten Voraussetzungen auch für CAN maximale Latenzzeiten garantieren lassen. Allerdings ergeben sich dabei relativ

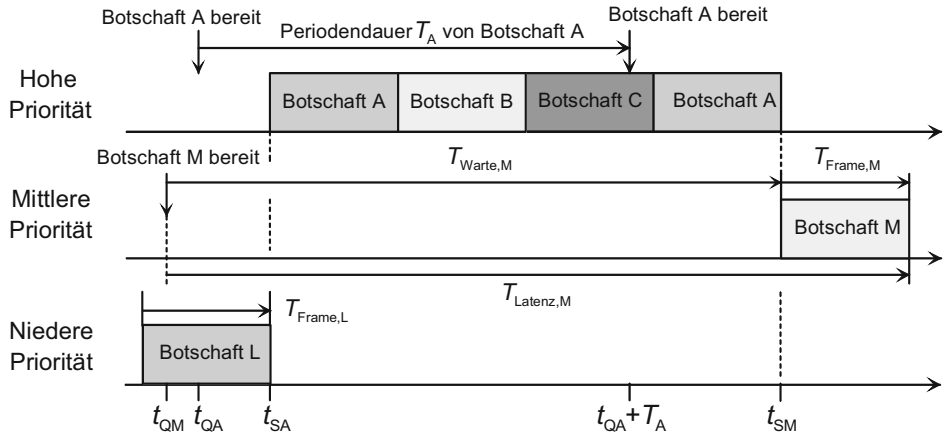


**Abb. 3.7** Szenario zur Bestimmung der Worst Case Latenzzeit

große Werte bzw. nur kleine zulässige Busauslastungen, so dass sich CAN für schnelle Echtzeitregelungen nicht so sehr wegen seiner theoretischen Defizite sondern vor allem wegen seiner relativ niedrigen Bitraten nur eingeschränkt eignet.

#### 3.1.7.1 Berechnung der maximalen Latenzzeiten

Abbildung 3.7 zeigt das Szenario, für das die maximalen Latenzzeiten bestimmt werden sollen. Eine Botschaft $M$ mit mittlerer Priorität wird zum Zeitpunkt $t = t_{QM}$ in den Sendepuffer (Sendewarteschlange, engl. *Queue*) gestellt. Zu diesem Zeitpunkt sei der Bus aber gerade durch eine andere Botschaft $L$ belegt. Da eine laufende Übertragung bei CAN nicht abgebrochen wird, wird diese Botschaft unabhängig von ihrer Priorität in jedem Fall vollständig übertragen. Botschaft $M$ muss also mindestens bis zum Zeitpunkt $t = t_{SA}$ warten. In der Zwischenzeit seien nun aber auch weitere Botschaften $A$, $B$ und $C$ zum Senden bereitgestellt worden, die eine höhere Priorität als Botschaft $M$ aufweisen sollen. Auch auf diese Botschaften muss Botschaft $M$ warten. Im Beispiel ist die Wartezeit so groß, dass die Botschaft $A$, die mit kurzer Periodendauer $T_A$ zyklisch gesendet wird, während der Wartezeit von Botschaft $M$ sogar zweimal übertragen wird. Erst zum Zeitpunkt $t = t_{SM}$ kann Botschaft $M$ dann endlich gesendet werden.

Die Voraussetzungen für eine Berechnung der Worst Case Latenzzeit sind [4, 5]:

- Die *Message Identifier* und damit die Prioritäten aller Botschaften des CAN Bussystems sowie deren Länge $T_{Frame,k}$ müssen bekannt sein. LP($M$) sei die Menge aller Botschaften, die eine niedrigere Priorität haben als die betrachtete Botschaft $M$. HP($M$) sei die Menge aller Botschaften, die eine höhere Priorität haben.
- Für alle Botschaften, die zyklisch gesendet werden, muss deren jeweilige Sendeperiode $T_k$ bekannt sein. Für Botschaften, die nicht zyklisch gesendet werden, wird $T_k$ als der

Mindestabstand zwischen zwei Sendeversuchen interpretiert (*Interarrival Time*) und muss ebenfalls bekannt sein.
- Alle Übertragungen erfolgen streng gemäß der Priorität der Botschaften. Dies muss auch für die Verarbeitung der Botschaften in den Sende- und Empfangspuffern der CAN Kommunikationscontroller gelten. Bei FIFO-artig organisierten Puffern in einfacheren Kommunikationscontrollern galt dies in der Vergangenheit leider nicht immer.
- Keine Übertragungsfehler und daher auch keine automatische Wiederholung.

Nicht berücksichtigt wird in den folgenden Betrachtungen, dass die Botschaftsdaten durch die Anwendungssoftware des Steuergerätes bereitgestellt werden und der Bereitstellungszeitpunkt damit ebenfalls mehr oder weniger schwankt. Auf das Zeitverhalten typischer Steuergeräte-Betriebssysteme wird in Kap. 7 eingegangen.

Unter den genannten Voraussetzungen ergibt sich die Wartezeit der Botschaft $M$ zu:

$$T_{\text{Warte},M} = \max_{k \in \text{LP}(M)} \left( T_{\text{Frame},k} \right) + \sum_{k \in \text{HP}(M)} \left\lceil \frac{T_{\text{Warte},M}}{T_k} \right\rceil \cdot T_{\text{Frame},k}. \tag{3.6}$$

Dabei beschreibt der erste Term die Wartezeit auf die längste Botschaft mit niedrigerer Priorität. Der Summenterm gibt die Wartezeit auf die Botschaften mit einer höheren Priorität an. Der Faktor $T_{\text{Warte},M} / T_k$, der auf die nächste ganze Zahl aufgerundet werden muss, berücksichtigt, dass die Wartezeit der Botschaft $M$ länger werden kann als die Periodendauer einer höherprioren Botschaft, so dass diese während der Wartezeit mehrfach gesendet wird. Da $T_{\text{Warte},M}$ auf beiden Seiten der Gleichung auftaucht, kann das Ergebnis leider nicht direkt, sondern nur durch Iteration bestimmt werden. Als Anfangswert der Iteration setzt man typischerweise den Summenterm zu 0. Die Iteration konvergiert, falls die mittlere Buslast

$$\text{BL} = \sum_{\text{alle } k} \frac{T_{\text{Frame},k}}{T_k} < 100\,\% \tag{3.7}$$

ist. Die maximale Latenz der Botschaft $M$ ergibt sich dann zu

$$T_{\text{Latenz},M,\text{max}} = T_{\text{Warte},M} + T_{\text{Frame},M} < T_{\text{D}} < T_M. \tag{3.8}$$

Bei einer Echtzeitanwendung wird in der Regel eine Obergrenze $T_{\text{D}}$ *(Deadline)* für die Latenzzeit jeder Botschaft vorgegeben. Sinnvollerweise ist die Obergrenze kleiner als die Periodendauer $T_M$ der jeweiligen Botschaft. Wird diese Bedingung nicht eingehalten, geht die Botschaft eventuell verloren. Bei den meisten Kommunikationscontrollern wird nämlich der Sendepuffer überschrieben, wenn eine weitere Botschaft mit demselben *Message Identifier* in den Puffer geschrieben wird, bevor die vorige Botschaft gesendet wurde. Die Schwankungsbreite der Latenzzeit ist der Jitter

$$T_{\text{Jitter},M} = T_{\text{Latenz},M,\text{max}} - T_{\text{Latenz},M,\text{min}}. \tag{3.9}$$

**Tab. 3.4** Satz von CAN Botschaften ($f_{\text{Bit}} = 125$ kbit/s, Worst Case Stuffing)

| CAN ID | $n_{\text{Data}}$ | Period $T$ | $T_{\text{Latency,min}} = T_{\text{Frame}}$ | $T_{\text{Latency, max}} = T_{\text{Warte,max}} + T_{\text{Frame}}$ |
|---|---|---|---|---|
| 1 (highest priority) | 1 byte | 50 ms | 0,5 ms | 1,4 ms |
| 2 | 2 byte | 5 ms | 0,6 ms | 2,0 ms |
| 3 | 1 byte | 5 ms | 0,5 ms | 2,6 ms |
| 4 | 2 byte | 5 ms | 0,6 ms | 3,2 ms |
| 5 | 1 byte | 5 ms | 0,5 ms | 3,7 ms |
| 6 | 2 byte | 5 ms | 0,6 ms | 4,3 ms |
| 7 | 6 byte | 10 ms | 0,9 ms | 5,0 ms |
| 8 | 1 byte | 10 ms | 0,5 ms | 8,6 ms |
| 9 | 2 byte | 10 ms | 0,6 ms | 9,2 ms |
| 10 (lowest priority) | 3 byte | 10 ms | 0,7 ms | 9,9 ms |

Tabelle 3.4 zeigt beispielhaft einen Satz von CAN Botschaften und die nach den obigen Formeln ermittelten Latenzzeiten. Alle Botschaften können innerhalb der geforderten Periodendauer gesendet werden. Allerdings haben die Botschaften mit den CAN IDs 6, 9 und 10 wegen der relativ hohen Busauslastung BL von etwa 85 % nur einen sehr geringen Sicherheitsabstand. Die Busauslastung und damit die maximalen Latenzzeiten könnten verringert werden, wenn mehrere Botschaften mit wenigen Nutzdatenbytes zu einer größeren Botschaft zusammengefasst würden. Naturgemäß ist dies nur bei Botschaften desselben Senders möglich.

#### 3.1.7.2 Festlegung der Botschaftsprioritäten

Das Beispiel zeigt erwartungsgemäß, dass die maximale Latenzzeit sowie der Jitter massiv von der Priorität der Botschaft abhängen. Als Empfehlung für die Botschaftsprioritäten, d. h. die *Message Identifier*, gilt daher, dass die Priorität umso größer, d. h. der *Message Identifier* umso kleiner gewählt werden sollte, je kleiner die Periodendauer bzw. je kürzer die *Deadline* der Botschaft ist (*Rate Monotonic Priority* bzw. *Deadline Monotonic Priority*) [4]. Diese Faustregel ist einfach anzuwenden, in Grenzfällen aber möglicherweise nicht optimal. Sollten einzelne Botschaften die geforderten Latenzzeiten nicht erreichen, kann mit dem in [5] beschriebenen iterativen Verfahren nach Audsley überprüft werden, ob tatsächlich keine Prioritätsverteilung existiert, mit der die Anforderungen eingehalten werden können.

Durch Übertragungsfehler vergrößern sich die Latenzzeiten. Wenn ein Fehler festgestellt wird, sendet der Kommunikationscontroller einen Error Frame mit 31 bit und der Sender wiederholt die fehlerhafte Botschaft. Dies kann modelliert werden, indem man auf der rechten Seite von Gl. 3.6 bei der Iteration den folgenden Term addiert:

$$E\left(T_{\text{Warte},M} + T_{\text{Frame},M}\right) = \left[31T_{\text{bit}} + \max_{k \in \text{HP}(M) \cup M}\left(T_{\text{Frame},k}\right)\right] \cdot \left\lceil \frac{T_{\text{Warte},M} + T_{\text{Frame},M}}{T_{\text{Error}}} \right\rceil. \quad (3.10)$$

Dabei wird angenommen, dass die Fehler mit der Fehlerrate $1 / T_{\text{Error}}$ auftreten. Komplexere Fehlermodelle wie bündelartige auftretende Fehler (*Burst Errors*) werden z. B. in [7] beschrieben.

#### 3.1.7.3 Festlegung von Botschaftsperiode und Phasenoffset

Die maximale Latenzzeit tritt dann auf, wenn alle Botschaften gleichzeitig sendebereit werden. Diese Situation kann vermieden werden, wenn für die Botschaftsperioden ganzzahlige Vielfache gewählt und feste Phasenbeziehung zwischen den einzelnen Botschaften festgelegt werden. Da der *Data Link Layer* bei CAN keine netzweite Synchronisation der Kommunikationscontroller bereitstellt, kann dies allerdings nur lokal für den Botschaftssatz eines einzelnen Busknotens erfolgen oder muss mit Hilfe von Synchronisationsbotschaften zwischen den Netzknoten durch die Anwendungssoftware sichergestellt werden [8]. Abbildung 3.8a zeigt das Problem, wenn zwei Botschaften nicht synchronisiert werden. Botschaft *A* werde alle $T_A = 5$ ms gesendet, Botschaft B alle $T_B = 8$ ms. Die Sequenz beginnt mit Botschaft A. Zufällig 3 ms später wird Botschaft B zum ersten Mal gesendet. Bei $t = 11$ ms sollte Botschaft B zum zweiten Mal gesendet werden. Da zu diesem Zeitpunkt aber Botschaft A wieder auf dem Bus liegt, verzögert sich Botschaft B. Dadurch verlängert sich diese Periode effektiv auf 9 ms, während sich die folgende Periode auf 7 ms verkürzt, weil Botschaft B zum Zeitpunkt $t = 19$ ms wieder planmäßig gesendet werden kann. Bei $t = 20$ ms liegt Botschaft B immer noch auf dem Bus, so dass die eigentlich anstehende Botschaft A erst bei $t = 21$ ms gesendet werden kann. Deren Periode verlängert sich dadurch auf 6 ms und wird beim nächsten planmäßigen Senden von Botschaft A auf 4 ms verkürzt. Im Beispiel *jittern* beide Periodendauern somit um ± 1 ms. Wesentlich günstiger ist es, wenn man die Periodendauern als ganzzahlige Vielfache, z. B. $T_A = 5$ ms und $T_B = 2\ T_A = 10$ ms wählt und die Botschaftssequenz mit einer Phasenverschiebung $T_A / 2 = 2{,}5$ ms startet. Da die Botschaften nun zu keinem Zeitpunkt kollidieren, sind die Periodendauern ohne Jitter konstant. Das Beispiel zeigt den positiven Einfluss von Phasenoffsets. Berücksichtigt man diese bei der Berechnung [9], so gibt es allerdings nicht nur ein einziges Worst-Case-Szenario, sondern es müssen sämtliche Kombinationen von Phasenlagen analysiert werden. Dies ist manuell sehr aufwendig, so dass in der Praxis Scheduling-Analysewerkzeuge eingesetzt werden, wie sie in Abschn. 9.8 beschrieben werden.

### 3.1.8 Time-Triggered-CAN (TTCAN) – Deterministischer Buszugriff

Aufgrund des CSMA/CR-Verfahrens kann nur für die Botschaften mit der höchsten Priorität eindeutig, für andere Botschaften aber nur unter bestimmten weiteren Voraussetzungen die maximale Latenzzeit garantiert werden, innerhalb der die Botschaft auch im ungünstigsten Fall noch sicher übertragen werden kann (siehe voriger Abschnitt). In jedem Fall aber kann es zu deutlich schwankenden Latenzzeiten, d. h. einem großen Jitter kommen, wenn die Steuergeräte zeitlich völlig unsynchronisiert auf den Bus zugreifen und es zu einer kurzzeitigen Belastungsspitze auf dem Bus kommt, weil alle Steuergeräte gleichzeitig

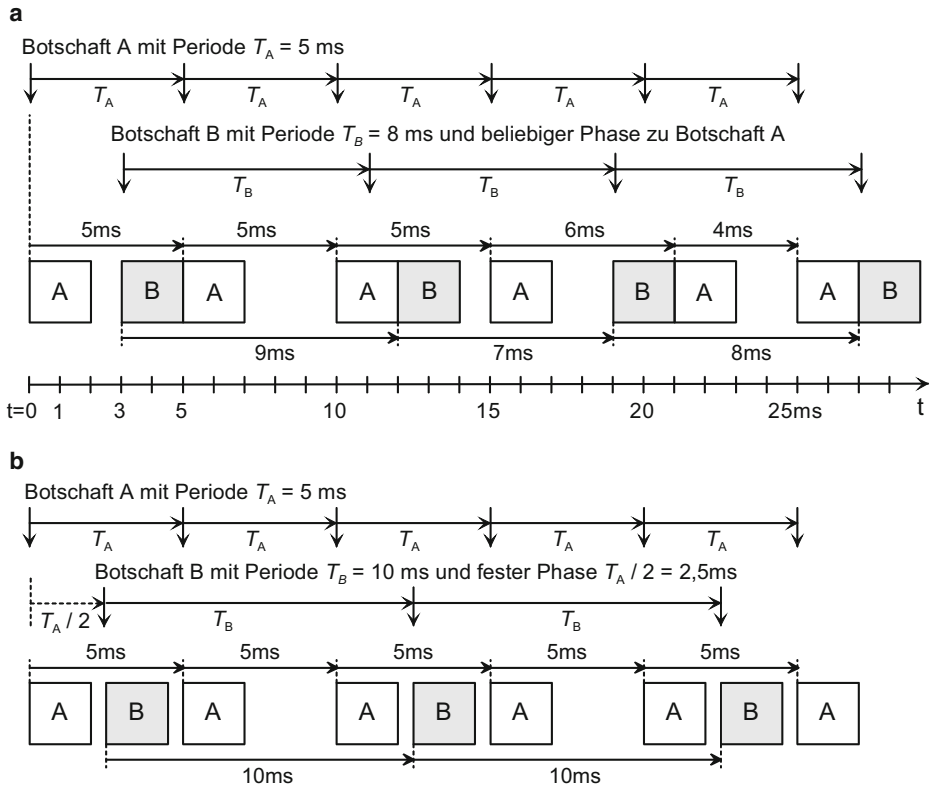


**Abb. 3.8** Einfluss der Botschaftsperioden und Phasen auf die Latenzzeit **a** beliebige Perioden und Phasen **b** Perioden als ganzzahlige Vielfache und feste Phasenbeziehungen

senden wollen. Mit Hilfe von Berechnungen und Simulationen lassen sich die maximalen Latenzzeiten und der Jitter in der Regel bestimmen. Allerdings sind bei komplexen Netzen mit sehr vielen Steuergeräten und Botschaften in der Praxis nicht alle Randbedingungen für alle Botschaften vollständig bekannt und oft stark von Implementierungsdetails in den Steuergeräten sowie von der aktuellen Konfiguration abhängig. Diese können sich durch Ausstattungsvarianten eines Fahrzeugs während eines Entwicklungsprojekts oder bei einem *Facelift* in der Serienproduktion in schwer vorhersehbarer Form ändern. CAN gilt daher als nicht im strengen Sinn deterministisch arbeitendes Bussystem, auch wenn es in der Praxis in der Regel bei weitem „gut genug" ist. CAN in seiner *klassischen* Art wird daher für Anwendungen mit extremen Sicherheitsforderungen wie Steer-by-Wire und Brake-by-Wire als nur bedingt geeignet betrachtet.

Um ein streng deterministisches Übertragungsverhalten zu garantieren, muss ein synchronisierter Buszugriff erfolgen, bei dem vorgegeben ist, welches Steuergerät in welchem Zeitfenster auf den Bus zugreifen darf (Time Division Multiple Access TDMA). Hierfür wird in ISO 11898-4 folgendes Konzept für CAN vorgeschlagen:

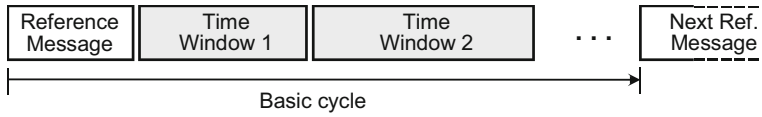


**Abb. 3.9** TTCAN Grundzyklus

Von einem Steuergerät, das als Zeit-Master arbeitet, wird periodisch eine Referenz-Botschaft ausgesendet (Abb. 3.9). Damit beginnt ein TTCAN-Grundzyklus (Basic Cycle). Alle anderen Steuergeräte synchronisieren sich durch diese Nachricht mit dem Grundzyklus. Der folgende Abschnitt des Grundzyklus enthält eine frei wählbare Anzahl von Zeitfenstern mit frei definierbarer, auch unterschiedlicher Länge. Es gibt drei Arten derartiger Zeitfenster:

- In den exklusiv reservierten Zeitfenstern darf jeweils genau ein Steuergerät seine Botschaften senden, d. h. diese Zeitfenster sind für die zeitsynchrone (time triggered) Kommunikation vorgesehen. In diesen Fenstern kann es nicht zu einer Kollision auf dem Bus kommen.
- In den arbitrierenden Zeitfenstern dürfen mehrere Steuergeräte ihre Botschaften senden, es wird das gewöhnliche CSMA/CR-Buszugriffsverfahren verwendet, d. h. ereignisgesteuerte (event triggered) Kommunikation. Ist das Zeitfenster lang genug, können mehrere Geräte hintereinander ihre Botschaften übertragen. Jeder Sender muss allerdings vor dem Senden überprüfen, ob seine Botschaft bis zum Ende des Fensters noch vollständig übertragen werden kann.
- Freie Zeitfenster werden für spätere Erweiterungen vorgesehen, so dass weitere Botschaften hinzugefügt werden können, ohne dass das Kommunikationsschema geändert werden muss.

Wenn eine Botschaft sehr oft übertragen werden soll oder wenn ein Steuergerät viele Botschaften zu übertragen hat, können einem Steuergerät auch mehrere Zeitfenster innerhalb eines Grundzyklus zugeordnet werden. Da manche Botschaften nicht in jedem Grundzyklus übertragen werden müssen, können mehrere Grundzyklen zu einem so genannten Systemzyklus (oder Matrixzyklus) zusammengefasst werden, wobei dann die Zeitfenster in den einzelnen Grundzyklen unterschiedlich belegt werden und sich der Gesamtablauf erst mit jedem Systemzyklus wiederholt.

Das gesamte System wird statisch konfiguriert, d. h. die Zeitfenster werden in der Entwicklungsphase und nicht zur Laufzeit festgelegt und den jeweiligen Steuergeräten zugeordnet. Ein einzelnes Steuergerät muss nicht das gesamte Kommunikationsschema kennen, sondern nur wissen, in welchen Zeitfenstern es seine eigenen Botschaften senden darf und weiß, in welchen Zeitfenstern die zu empfangenden Botschaften ankommen müssen.

Um einen Totalausfall bei Defekt des Zeit-Masters zu vermeiden, kann ein System bis zu acht Zeit-Master haben, deren Referenzbotschaft über die Wahl des CAN-Identifiers mit

unterschiedlicher Priorität versehen wird, so dass stets der Master mit der aktuell höchsten Priorität den Grundzyklus vorgibt.

Die Zeitablaufsteuerung kann mit gewöhnlichen CAN-Controllern vollständig in Software realisiert werden, wenn die Zeitfenster ausreichend groß definiert werden und der Zeitjitter akzeptabel ist, der durch die Zeitablaufsteuerung und das Bereitstellen der Sendedaten durch die Software entsteht. Problematisch sind dabei allerdings die automatische Sendewiederholung bei Übertragungsfehlern sowie die automatische Wiederholung von Botschaften, die in den arbitrierenden Zeitfenstern den Buszugriff „verloren" haben. Falls die Übertragungswiederholung nicht innerhalb des Zeitfensters abgeschlossen werden kann, muss die automatische Wiederholung abgeschaltet werden. Bei neueren CAN-Controllern ist dies in der Regel möglich.

Um die Mikrocontroller-Belastung durch eine rein softwaremäßige Umsetzung der Zeitablaufsteuerung zu vermeiden, stehen mittlerweile entsprechende CAN-Controller zur Verfügung, die das TTCAN-Protokoll weitgehend hardwaremäßig abwickeln und durch die Software lediglich konfiguriert werden müssen.

Die zeitsynchrone Kommunikation eignet sich besonders für Mess- und Regelungsaufgaben, die periodisch durchgeführt werden müssen. Um zusätzliche, insbesondere veränderliche Totzeiten zu verhindern, sollte dabei die Bearbeitung der Mess- und Regelaufgaben im Steuergerät zeitlich mit dem TTCAN-Grundzyklus synchronisiert werden. Für ereignisgesteuerte Botschaften, insbesondere solche mit hoher Priorität, ergibt sich bei TTCAN in der Regel eine größere Latenzzeit als beim normalen CAN, da nur ein Teil des Grundzyklus für solche Botschaften reserviert werden kann.

Für die übergeordnete Synchronisation im System definiert TTCAN auch ein Verfahren, bei dem eine globale Systemzeit übertragen wird und lokal innerhalb eines Grundzyklus eine Driftkorrektur des lokalen Zeitgebers erfolgt.

TTCAN gewährleistet zwar die deterministische Übertragung von Botschaften, erlaubt aber keine höheren Bitraten als der klassische CAN-Bus, da die bitweise Arbitrierung in den ereignisgesteuerten Zeitfenstern und die positive bzw. negative Empfangsbestätigung innerhalb des *Acknowledge Bits* am Ende jeder Botschaft beibehalten wird. Damit ist das eigentliche Grundproblem, die für anspruchsvolle Anwendungen eigentlich nicht mehr ausreichende Bandbreite, nicht gelöst. TTCAN hat sich daher bisher nicht durchgesetzt. CAN soll aus diesem Grund mittelfristig in den Anwendungsbereichen Fahrwerks- und Triebstrangelektronik durch FlexRay abgelöst werden. Mittlerweile kündigt sich aber mit CAN Flexible Datarate (siehe übernächster Abschnitt) eine interessante Alternative an.

### 3.1.9 Energiesparmaßnahmen: Wakeup und Partial Networking

Während die meisten Steuergeräte bei abgestelltem Motor ausgeschaltet werden können, müssen einzelne Systeme, z. B. die Zentralverriegelung, weiterhin betriebsfähig sein. Um die Batterieentladung zu verringern, werden aber auch diese Geräte in einen stromsparenden Zustand versetzt (*Sleep Mode*) und von Zeit zu Zeit *aufgeweckt*. Gemäß

ISO 11898-5 können die an einem CAN-Bus angeschlossenen Steuergeräte z. B. durch ein *CAN Wakeup Pattern* aufgeweckt werden. Als *Wakeup Pattern* dient dabei ein dominantes Bit auf dem CAN-Bus mit einer Dauer von mehr als 5 µs nach einem vorherigen rezessiven Ruhezustand. Um Störungen zu unterdrücken, werden Impulse ignoriert, die kürzer sind als 0,75µs. In der Praxis wird das *Wakeup Pattern* erzeugt, indem eine beliebige CAN-Botschaft versendet wird, die eine ausreichend lange Folge von 0 Bits enthält. Im Schlafzustand des Steuergerätes ist üblicherweise nur der CAN-Transceiver aktiv, der CAN-Kommunikationscontroller und die CPU sind abgeschaltet (Abb. 3.1). Nach dem Empfang des *Wakeup Patterns* dauert es daher einige Zeit, bis das Steuergerät tatsächlich betriebsfähig ist. Wie und wann der Übergang eines Steuergerätes und eines gesamten CAN-Bussystems vom aktiven zum stromsparenden Zustand und umgekehrt erfolgt, muss auf Anwendungsebene festgelegt werden. Zur Koordination der Abläufe dienen z. B. das OSEK bzw. AUTOSAR Netzmanagement (Abschn. 7.2.3 und 8.5) und der AUTOSAR ECU State Manager (Abschn. 8.2).

Mittlerweile hat man erkannt, dass es sinnvoll ist, wenn man nicht alle, sondern gezielt nur einzelne oder eine Gruppe von Geräten aktivieren kann (*Selective Wakeup*). Insbesondere denkt man daran, Steuergeräte nicht mehr nur bei abgestelltem Motor, sondern auch im Fahrbetrieb abzustellen, um auch dort Energie zu sparen. So macht es beispielsweise wenig Sinn, das Steuergerät der Einparkhilfe auch bei Autobahnfahrten aktiv zu halten.

Der Betrieb eines CAN-Systems, bei dem nur ein Teil der Steuergeräte aktiv ist, wird als Teilnetzbetrieb (*Partial Networking*) bezeichnet. Für den Teilnetzbetrieb wurde vorgeschlagen, nicht ein einfaches Bitmuster sondern eine komplette CAN-Botschaft, einen *Wakeup Frame*, zum Aufwecken zu verwenden. Um ein Steuergerät aufzuwecken, müssen der *CAN Message Identifier* und die Datenlänge (*Data Length Code DLC*) des *Wakeup Frames* mit den im Steuergerät vorkonfigurierten Werten übereinstimmen. Außerdem muss ein vordefiniertes Bit innerhalb des Datenfelds gesetzt sein. Dadurch lassen sich mit einer einzigen CAN-Botschaft gezielt bis zu 64 Steuergeräte selektiv aktivieren. Im Gegensatz zu einfachen *Wakeup Pattern*, die von den meisten heute erhältlichen *CAN Transceivern* erkannt werden können, ist die Erkennung einer vollständigen CAN-Botschaft und damit die entsprechenden *CAN Transceiver* erheblich aufwendiger. Da ein wesentlicher Teil der *Data Link Layer* Logik des CAN Kommunikationscontrollers im *Transceiver* dupliziert werden und eine Konfigurationsmöglichkeit des *Transceivers* durch den Mikrocontroller geschaffen werden muss, wird es einige Zeit dauern, bis die zugehörigen CAN-Standardisierungsarbeiten als ISO 11898-6 abgeschlossen und geeignete ICs auf dem Markt verfügbar sein werden, bevor dann die entsprechenden Funktionalitäten im Netzmanagement und auf Anwendungsebene definiert und umgesetzt werden können.

### 3.1.10 Höhere Datenraten: CAN Flexible Data-Rate CAN FD

Wegen seiner begrenzten Busbandbreite gilt CAN seit einiger Zeit als Flaschenhals für zukünftige Fahrwerks- und Triebstrangsysteme. Der als CAN-Nachfolger konzipierte Flex-

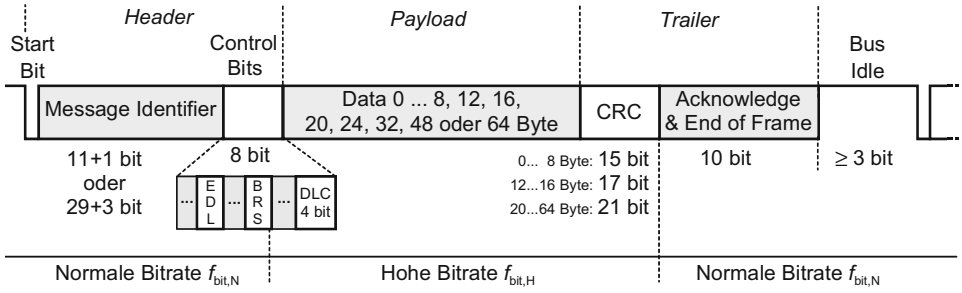


**Abb. 3.10** Botschaftsformat bei CAN FD (Längenangaben ohne Bit-Stuffing)

Ray Bus (siehe Abschn. 3.3) konnte bisher jedoch nicht alle Hoffnungen erfüllen. Zum einen steigt die nutzbare Busbandbreite wegen des gerade bei kurzen Botschaften verhältnismäßig ineffizienten FlexRay-Protokolls weniger stark als es die höhere Bitrate erwarten lässt. Zum anderen ist dessen TDMA-Konzept relativ starr, so dass das Kommunikationsschema in der Entwicklungsphase sehr sorgfältig geplant werden muss, weil nachträgliche Änderungen schwierig sind. Auch die Umstellung der Software vom ereignis- zum zeitgesteuerten Betrieb macht den Übergang vorhandener CAN-Systeme auf FlexRay in der Praxis aufwendig. Mit dem Bosch-Konzept *CAN with Flexible Data-Rate CAN FD*, dessen Spezifikation in Version 1.0 seit 2012 vorliegt, soll die Bandbreite von CAN Systemen in aufwärtskompatibler Weise gesteigert werden. Dies erfolgt durch zwei Maßnahmen:

- Längere Botschaften
  Die Nutzdatenlänge einer CAN-Botschaft wird von maximal 8 Byte auf 64 Byte erhöht. CAN FD Botschaften können damit wie bisher 0 bis 8 Byte oder nun auch 12, 16, 20, 24, 32, 48 oder 64 Byte Nutzdaten enthalten. Da die Längenangabe *Data Length Code DLC* innerhalb der *Control Bits* im Header (Abb. 3.4) schon bisher 4 bit groß war, müssen dazu lediglich die bisher ungenutzten Codes vergeben werden. Außerdem muss die CRC-Prüfsumme im Trailer angepasst werden, um bei längeren Botschaften dieselbe Datensicherheit zu gewährleisten. Botschaften mit bis zu 8 Byte Nutzdaten verwenden weiterhin die bisherige 15 bit CRC. Für 12 und 16 Byte verlängert sich die Prüfsumme auf 17 bit, ab 20 Byte wird eine 21 bit CRC verwendet. Um eine CAN FD Botschaft von einer normalen CAN-Botschaft unterscheiden zu können, wird im Header das neue *Extended Data Length EDL* Bit eingeführt (Abb. 3.10).
- Höherer Bittakt
  Die begrenzenden Faktoren für die Bitrate bei CAN sind die bitweise Arbitrierung des *Message Identifiers* im Botschaftsheader zur Kollisionserkennung sowie das *Acknowledge Bit* zur Bestätigung des korrekten Empfangs im Botschaftstrailer. Die Auswertung dieser Bits muss in allen Steuergeräten des Busses zuverlässig innerhalb desselben Bittaktes erfolgen. Weil die Signallaufzeit auf den Busleitungen unveränderbar ist, entsteht dadurch die in Gl. 3.1 beschriebene Abhängigkeit zwischen Buslänge und minimaler Bitdauer

**Tab. 3.5** Abschätzung der Nutzdatenraten (für 11 bit *Message Identifier*)

| $n_{\text{Data}}$ | Klassischer CAN 2.0 $f_{\text{bit}} = 500$ kbit/s | CAN FD ohne Bitratenumschaltung $f_{\text{bit,N}} = f_{\text{bit,H}} = 500$ kbit/s | CAN FD mit Bitratenumschaltung $f_{\text{bit,N}} = 500$ kbit/s, $f_{\text{bit,H}} = 4$ Mbit/s |
|---|---|---|---|
| 8 Byte | Nicht möglich | 29 KB/s | 79 KB/s |
| 16 Byte | | 35 KB/s | 131 KB/s |
| 32 Byte | | 40 KB/s | 195 KB/s |
| 64 Byte | | 44 KB/s | 260 KB/s |

bzw. maximaler Bitrate. Für das eigentliche Nutzdatenfeld und die CRC-Prüfsumme dagegen gilt diese Forderung nicht unbedingt. CAN FD schlägt daher vor, die Bitrate in diesem Teil der Botschaft deutlich zu erhöhen. Die vorliegende Spezifikation 1.0 nennt allerdings keine konkreten Zahlenwerte. In begleitenden Veröffentlichungen von Bosch [8] werden für den aus ISO 11898-2 bekannten Physical Layer Zielwerte um 4 Mbit/s genannt. Die Bitratenumschaltung ist optional und wird daher durch das neu eingeführte Steuerbit *Bit Rate Switch BRS* im *Control Feld* des Headers signalisiert (Abb. 3.10). Mit dem Ende des CRC-Felds im Trailer wird auf die normale Bitrate zurückgeschaltet.

Der Header einer CAN FD-Botschaft verlängert sich um die neu eingeführten EDL und BRS Bits. Der Trailer vergrößert sich durch die modifizierte CRC-Prüfsumme je nach Nutzdatenlänge um maximal 6 bit. Analog zu Gl. 3.4 lässt sich die Dauer einer Botschaft abschätzen zu

$$T_{\text{Frame}} < 1,25 \cdot \left[ 29\,\text{bit}\,(49\,\text{bit}) \cdot \frac{1}{f_{\text{bit},N}} + \{26\,\text{bit} + n_{\text{Data}}\} \cdot \frac{1}{f_{\text{bit},H}} \right] \tag{3.11}$$

wobei $n_{\text{Data}}$ die Anzahl der Datenbits darstellt und der Faktor 1,25 wieder das *Bit Stuffing* im ungünstigsten Fall berücksichtigt. Der Wert in (…) gilt für 29 bit *Message Identifier*. Die Nutzdatenrate ist wieder $f_{\text{Data}} = n_{\text{Data}} / T_{\text{Frame}}$. Wie man an den Daten von Tab. 3.5 sieht, verspricht CAN FD tatsächlich eine massive Vergrößerung der Nutzdatenrate, falls die Umschaltung auf hohe Bitraten zuverlässig funktioniert.

### 3.1.11 Zusammenfassung CAN – Layer 1 und 2

- Kommunikation zwischen mehreren Kfz-Steuergeräten zum Austausch von Mess-, Steuer- und Regelsignalen im Echtzeitbetrieb mit hoher Fehlersicherheit.
- Bitstrom-orientiertes Übertragungsprotokoll mit bidirektionaler Zwei-Draht-Leitung als Linien-Bus. Buslänge und zulässige Länge der Stichleitungen abhängig von der Bitrate, max. 1 Mbit/s bei < 40 m Buslänge und Stichleitungen < 30 cm.

- Übliche Bitraten im Kfz 500 kbit/s (High Speed CAN) und 100 … 125 kbit/s (Low Speed CAN), andere Bitraten möglich.
- Botschaften mit 0 … 8 Datenbytes und 6 … 8 Byte Overhead (Header/Trailer)
- CAN-Controller, Transceiver und Mikroprozessor notwendig.
- Broadcast-System, bei dem die Nachrichten durch Message Identifier gekennzeichnet und beim Empfänger davon abhängig Empfangsentscheidungen (Akzeptanzfilterung) getroffen werden. Die Message Identifier priorisieren gleichzeitig Botschaften beim Sendeversuch (Buszugriff mit CSMA/CR). Message Identifier mit 11 bit und 29 bit Länge (CAN 2.0A und 2.0B), können im selben CAN-Netz gemischt verwendet werden.
- CAN-Controller mit Hardware-Akzeptanz-Filterung (Full CAN) und Software-Akzeptanz-Filterung (Basic CAN) erhältlich.
- Netzweite Datenkonsistenz, da alle CAN-Controller die empfangenen Daten ignorieren, wenn ein oder mehrere Geräte einen Übertragungsfehler erkennen. Automatische Übertragungswiederholung im Fehlerfall. Automatische Abschaltung von defekten CAN-Controllern.
- Übertragungsdauer einer Botschaft mit 8 Datenbytes bei 500 kbit/s unter Berücksichtigung des *Bit-Stuffings* im ungünstigsten Fall:

  $$270\,\mu s\,(11\,\text{bit Identifier}),\ 320\,\mu s\,(29\,\text{bit Identifier}).$$

  Entspricht der Worst Case Latenz für die Botschaft mit der höchsten Priorität.
- Theoretisch mögliche Nutzdatenrate 29 KB/s (11 bit Identifier), 24 KB/s (29 bit Identifier)
- Verfahren zur Vergrößerung der Nutzdatenrate durch bis zu 64 Byte je Botschaft und dynamische Umschaltung auf höhere Bitraten in Entwicklung.

## 3.2 Local Interconnect Network LIN

LIN (Local Interconnect Network) ist ein relativ junges Bussystem, das Ende der 90er Jahre entwickelt wurde, um eine kostengünstige Alternative zu Low-Speed-CAN-Bussystemen für einfache Sensor-Aktor-Anwendungen zu bieten, z. B. für die Tür-, Sitz- oder Schiebedach-Elektronik [10]. Triebfeder war der Halbleiterhersteller Motorola (heutiger Name: Freescale) in Zusammenarbeit mit verschiedenen Kfz-Herstellern, die sich zu einem LIN-Konsortium zusammengeschlossen haben. Die Spezifikationen sind offen gelegt und da ein einfaches, zeichenorientiertes Protokoll verwendet wird, das mit jedem UART implementiert werden kann, ist es praktisch frei verfügbar.

### 3.2.1 Überblick

Der Physical Layer und die Bitübertragungsschicht (8N1 UART-Zeichen mit 8 Datenbits und je einem Start- und Stoppbit ohne Parität) entsprechen dem K-Line-Protokoll (siehe

Abschn. 2.2). Die mögliche Bitrate liegt im Bereich 1 … 20 kbit/s, wobei die drei Bitraten 2,4 kbit/s, 9,6 kbit/s bzw. 19,2 kbit/s empfohlen werden. In der ersten LIN-Generation (LIN Spezifikation V1.3) lag der Schwerpunkt darauf, möglichst kostengünstige Netzknoten zu ermöglichen. Um LIN schnell in den Markt zu bringen, wurde es als Sub-Bus für CAN-Netze positioniert, bei dem ein Master-Knoten (in der Regel gleichzeitig) als Gateway zu einem übergeordneten CAN-Netz fungiert und als einziger Knoten über eine präzise Zeitbasis verfügen und sich um die gesamte Netzkonfiguration kümmern muss. Sämtliche anderen Netzknoten sind Slave-Knoten, deren Bittakt selbstsynchronisierend ausgeführt werden kann und die keinerlei Konfigurationsinformationen über das Gesamtnetzwerk benötigen. Auf der Bussystem-Ebene verfügt LIN nur über wenige Mechanismen, um Übertragungsfehler zu erkennen, und über keinerlei vorgegebene Verfahren, derartige Fehler selbstständig zu korrigieren.

Mit der Version V2.0 der LIN-Spezifikation wurde das Protokoll nicht nur in einigen Teilen präziser spezifiziert, sondern auch um verschiedene, teilweise optionale Mechanismen erweitert, die die Geräte erheblich komplexer machen. Dazu gehört das optionale Tunneln von KWP 2000 oder UDS-Diagnosebotschaften, was dann aber die Implementierung von ISO-Diagnosefunktionen in den LIN-Steuergeräten erfordert, sowie ein zwingend geforderter *Plug-and-Play*-Mechanismus, der die automatische Konfiguration von LIN-Slave-Steuergeräten durch das LIN-Master-Steuergerät erlaubt. Der Mechanismus ist aber nur mit Steuergeräten sinnvoll, die über Flash-ROM- bzw. EEPROM-Speicher verfügen und wurde teilweise kritisiert, weil dies kostentreibend wirkt und damit eigentlich der LIN-Grundidee widerspricht.

Zusätzlich sind die Änderungen nicht vollständig rückwärtskompatibel. So wurde u. a. die Art der Prüfsummenberechnung verändert. Ein V2.0 Master-Steuergerät muss mit V1.3 Slave-Steuergeräten zusammenarbeiten können, während ein V1.3 Master-Steuergerät mit LIN V2.0 Slave-Steuergeräten nicht kommunizieren kann. Mit LIN V2.1 und V2.2 wurden einige Details weitgehend aufwärts kompatibel ergänzt, insbesondere aber einige Unklarheiten aus den Vorgängerversionen beseitigt.

Eine auf Betreiben von GM und Ford leicht modifizierte Variante von LIN V2.0, z. B. Bitrate max. 10,4 kbit/s, wurde als SAE J2602 zur Standardisierung eingereicht. Darüber hinaus gibt es proprietäre LIN-Varianten z. B. bei Toyota oder den sogenannten *Cooling Bus* der Hersteller von Klimaanlagen.

Nach anfänglicher Euphorie zeigte sich, dass das ursprüngliche Ziel, LIN-Knoten zur Hälfte der Kosten eines Low-Speed-CAN-Knotens zu implementieren, nur unter größten Anstrengungen zu erreichen ist. Dies liegt u. a. daran, dass CAN-Bustransceiver aufgrund der riesigen Stückzahlen nur unwesentlich teurer sind als K-Line-Bustransceiver. Auch LIN-Transceiver sind wegen der notwendigen Spannungsfestigkeit lediglich bei wenigen Halbleiterprozessen direkt auf dem Mikrocontroller integrierbar. Wenn die extrem niedrigen Bitraten von LIN tatsächlich ausreichen, können auch Low-Speed-CAN-Netze mit Eindraht-Leitungen realisiert werden. Und schließlich sind die Mehrkosten von CAN-Controllern gegenüber UARTs bei direkter Integration im Mikrocontroller und den heutigen Integrationsdichten fast vernachlässigbar. Umgekehrt macht die notwendige CAN-

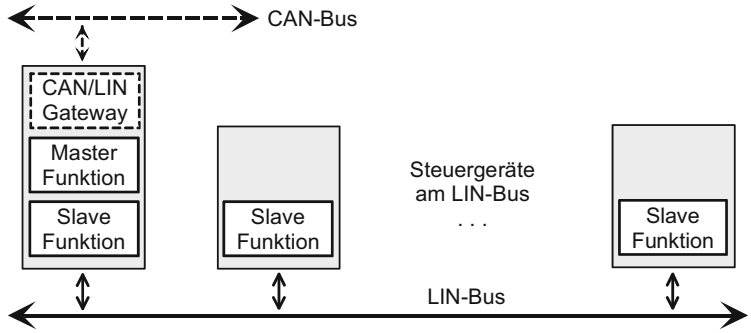


**Abb. 3.11** Struktur eines LIN-Bussystems

LIN-Gateway-Funktionalität das Gesamtnetz komplexer und damit auch fehleranfälliger. Trotzdem sind LIN-Busse in der Karosserieelektronik, etwa bei Tür-, Spiegel- und Fensterhebersteuerungen oder Multi-Funktions-Lenkrädern und Innen- und Außenlichtsteuergeräten, mittlerweile weit verbreitet. Die weitere Standardisierung erfolgt inzwischen im Rahmen von ISO 17987.

### 3.2.2 Data Link Layer

In LIN-Bussystemen, in den LIN-Spezifikationen als *LIN Cluster* bezeichnet, steuert genau ein fest definiertes Steuergerät als *Master* durch Senden von Botschaftsheadern den gesamten Kommunikationsablauf, worauf eines der Steuergeräte als *Slave* mit einer Daten-Botschaft antwortet. Dabei soll auch im Master-Steuergerät das Senden der Daten-Botschaften vom Senden der Header-Botschaften in der Protokollsoftware logisch getrennt werden, so dass das Senden von Daten in allen Steuergeräten grundsätzlich als Slave-Funktionalität realisiert wird (Abb. 3.11). Das Master-Steuergerät übernimmt dabei in der Regel auch die Gateway-Funktion zum übergeordneten CAN-Bus.

Das Master-Steuergerät sendet einen Botschaftsheader aus (Abb. 3.12), der mit mindestens 13 Low-Bits und 1 High-Bit beginnt, von denen der Slave mindestens 11 Low-Bits erkennen muss, nachdem der Bus vorher im Ruhezustand (*Bus Idle*) war. Diese als *Sync Break* bezeichnete Bitfolge ist das einzige nicht Standard-UART gemäße Zeichen und daher von jedem Empfänger zu jedem Zeitpunkt eindeutig als Botschaftsbeginn erkennbar. Danach folgt ein *Sync Byte* mit einer alternierenden Low-High-Bitfolge (55h), die eine Bittakt-Synchronisation der Empfänger erlaubt. Als letztes Zeichen des Headers sendet der Master das *Identifier Byte*. Mit dem LIN-Identifier wird wie bei CAN der Inhalt der folgenden Botschaft gekennzeichnet. Genau ein Steuergerät ist so konfiguriert, dass es auf diesen LIN-Identifier mit einer Daten-Botschaft mit 1 bis 8 Datenbyte und einem Prüfsummenbyte antwortet (*Unconditional Frames*).

Wie CAN arbeitet auch LIN verbindungslos. Der LIN-Identifier kennzeichnet den Inhalt der gewünschten Daten-Botschaft und nicht eine Stationsadresse (inhaltsbezogene

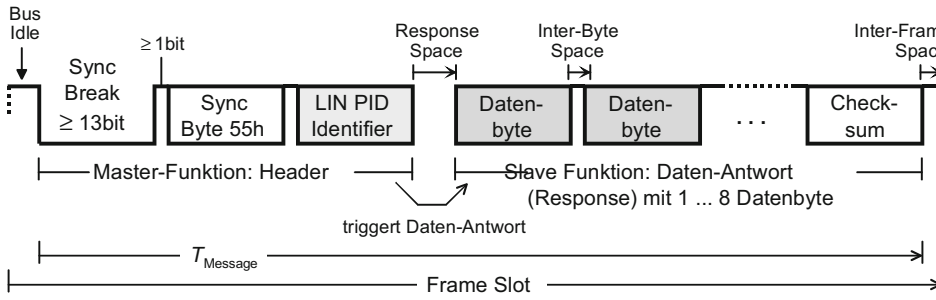


**Abb. 3.12** LIN-Botschaftsformat

**Tab. 3.6** LIN-Identifier (ohne Paritätsbits)

| Identifier (6 bit) | Bedeutung |
|---|---|
| 00h … 3Bh | Beliebige LIN-Botschaften, Inhalt durch LIN-Konfigurationsdatei festgelegt |
| 3Ch | Master-Request-Frame (MRF) für Transportprotokoll-Botschaften (Diagnose und Konfiguration, siehe Abschn. 3.2.5)<br>1. Datenbyte 00h: Befehl zum Umschalten in den *Sleep Modus*<br>1. Datenbyte ungleich 00h: Diagnosebotschaft nach Abb. 3.13 |
| 3Dh | Slave-Response-Frame (SRF) für Transportprotokoll-Botschaften |
| 3Eh, 3Fh | Reserviert für Erweiterungen |

Adressierung). Alle am Bus angeschlossenen Steuergeräte müssen die Header empfangen und auswerten und können alle Daten-Botschaften mithören. Der Identifier besteht aus 6 bit, mit denen der Inhalt der folgenden Daten-Botschaft gekennzeichnet wird, sowie 2 Paritätsbits, mit denen der Identifier gegen Übertragungsfehler gesichert wird. Das aus dem eigentlichen Identifier und den beiden Paritätsbits bestehende Byte wird in der LIN Spezifikation als *Protected Identifier PID* bezeichnet. Bei LIN 1.x wurde mit dem Identifier auch die Länge der Botschaft codiert, dabei waren 32 Identifier für Botschaften mit je 2 Datenbyte, 16 mit jeweils 4 Datenbyte und die übrigen für Botschaften mit je 8 Datenbyte vorgesehen. Bei LIN 2.x wird die Datenlänge in der Konfigurationsdatei des Systems (siehe Abschn. 3.2.6) unabhängig vom Wert des Identifiers beliebig zwischen 1 und 8 Datenbyte festgelegt. Für die Dateninhalte selbst gibt es in der LIN-Spezifikation keine Festlegung. Die Identifier 3Ch und 3Dh sind für Kommando- bzw. Diagnose-Botschaften (*Diagnostic Frames*) reserviert (Tab. 3.6), die die nachfolgend beschriebene Transportschicht des LIN-Protokolls verwenden. Die Identifier 3Eh und 3Fh sind für zukünftige LIN-Protokollerweiterungen vorgesehen (*Extended Frames*). (Hinweis: Die für die Identifier hier und im LIN-Standard angegebenen Werte verstehen sich jeweils ohne die beiden Paritätsbits).

Die Daten-Antwort des Slaves wird durch eine Prüfsumme abgesichert. Bei LIN V1.3 umfasst die Prüfsumme nur die Datenbytes (*Classic Checksum*). LIN V2.0 Slaves dagegen müssen in die Berechnung der Prüfsumme zusätzlich noch den Identifier aus dem zugehörigen Header mit einbeziehen (*Enhanced Checksum*). Die einzelnen Bytes werden wie bei

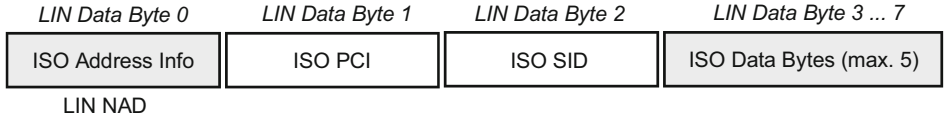


**Abb. 3.13** Abbildung einer ISO 15765-2 Single Frame Diagnosebotschaft auf LIN

UARTs üblich von einem Startbit und einem Stoppbit eingerahmt und mit dem LSB zuerst übertragen. Die Übertragung von Mehr-Byte-Daten erfolgt in *Little-Endian*-Reihenfolge, d. h. das niederwertige Byte wird zuerst gesendet.

Die Fehlererkennung und -behandlung ist bei LIN nur rudimentär definiert. Der Sender einer Botschaft erhält weder positive Empfangsbestätigungen noch Fehlermeldungen von den Empfängern. Der jeweilige Sender muss das von ihm gesendete Bussignal mitlesen. Stellt er einen Bitfehler auf der Busleitung fest, bricht er den Sendevorgang ab. Der Master muss erkennen, wenn kein Slave auf eine Header-Botschaft antwortet oder falls bei der Daten-Antwort eine Zeitüberschreitung auftritt. Die Slaves ignorieren Botschaften mit unbekanntem Identifier oder mit fehlerhafter Identifier-Parität sowie Daten-Botschaften mit fehlerhafter Prüfsumme. Festgestellte Fehler werden in der lokalen LIN-Protokoll-Software als Statusinformation gespeichert und können von der Applikationsschicht des jeweiligen Busteilnehmers abgefragt werden (siehe Abschn. 3.2.8). Die weitere Fehlerreaktion der Applikationsschicht ist anwendungsabhängig und in der LIN-Spezifikation nicht definiert. Mit LIN V2.0 neu eingeführt wurde, dass jeder LIN-Slave mindestens ein Status-Bit (*Response Error Bit*) bereitstellen muss, das in einer der periodisch an den Master gesendeten Botschaften enthalten sein muss. Das *Response Error Bit* soll gesetzt werden, wenn der Slave beim Empfang einer Botschaft oder beim Senden einer eigenen Botschaft einen Fehler festgestellt hat. Nach dem Versenden der Botschaft mit dem *Response Error Bit* setzt der Slave das Bit selbstständig zurück. Der Master soll die *Response Error Bits* aller Slave sowie seine eigene Fehlerüberwachung auswerten und in seiner Applikationsschicht geeignet reagieren. Mit welcher Periodendauer die Statusinformation gesendet werden muss und wie der Master reagieren soll, bleibt weiter anwendungsabhängig.

Mit dem Identifier 3Ch und 00h als erstem Datenbyte kann der Master alle Slaves in den so genannten *Sleep Modus* schalten, in dem alle Busaktivitäten eingestellt werden. Ein Slave aktiviert den *Sleep Modus* außerdem selbstständig, wenn der Bus für mindestens 25.000 Bitzeiten (bei LIN V2.0 frühestens nach 4 s und spätestens nach 10 s) inaktiv war. Jeder Busteilnehmer kann den *Sleep Modus* des Busses beenden, indem er eine *Wake Up* Botschaft sendet. Eine *Wake Up* Botschaft besteht aus einem einzelnen 80h Zeichen (ohne Header und Checksumme). Bei LIN V2.0 wurde dies geändert in ein *Low*-Signal für 0,25 … 5 ms. Alle angeschlossenen Busteilnehmer müssen die *Wake Up* Botschaft erkennen und sich innerhalb von 100 ms neu initialisieren. Der Master beginnt nach mindestens 4 und höchstens 64 Bitzeiten (bei LIN V2.0 nach 100 ms) wieder mit dem Aussenden von Header-Botschaften. Falls dies unterbleibt, wiederholt das Steuergerät die *Wake Up* Bot-

schaft bis zu drei Mal im Abstand von 150 … 250 ms. Bei Misserfolg kann dieser Ablauf im Abstand von min. 1,5 s wiederholt werden.

Das Timing der LIN-Botschaften ist sehr großzügig spezifiziert. Der Bittakt des Masters soll nicht mehr als ± 0,5 % vom Nominalwert abweichen. Der Bittakt der Slaves dagegen darf vor der Synchronisation um bis zu ± 15 % abweichen, wenn er durch das *Sync Byte* auf ± 2 % genau nachsynchronisiert wird und diese Toleranz bis zum Ende der Botschaft nicht überschreitet. Der Bittakt von Slaves, die sich nicht automatisch nachsynchronisieren, darf um ± 1,5 % vom Nominalwert abweichen. Für die Dauer des *Sync Break* ist lediglich eine Minimallänge von 14 bit vorgegeben und die Pause zwischen *Sync Byte* und *Identifier Byte* ist beliebig, solange die vorgegebene Maximallänge von 49 Bitzeiten nicht überschritten wird. Ähnlich freizügig wählbar sind der Abstand zwischen dem Ende des Headers und dem Beginn der Daten-Antwort (*Response Space*) sowie die maximal zulässige Pause zwischen den einzelnen Bytes der Daten-Antwort (*Inter Byte Space*). Einzelheiten zu den Zeitbedingungen werden im Abschn. 3.2.8 erläutert.

### 3.2.3 Neue Botschaftstypen bei LIN V2.0

Neben den ständig periodisch zu sendenden LIN-Botschaften (*Unconditional Frames*) gibt es seit LIN V2.0 auch *dynamische Botschaften*, die als *Sporadic Frames* und *Event Triggered Frames* bezeichnet werden. Wie für normale Botschaften müssen auch für diese Botschaften entsprechende Zeitschlitze in der Botschaftstabelle reserviert werden.

*Sporadic Frames* werden aber nur dann versendet, wenn tatsächlich Daten im Master-Steuergerät vorliegen oder vom Master-Steuergerät Slave-Antworten gefordert werden, ansonsten erzeugt der Master keine Header-Botschaft und der Bus bleibt während des entsprechenden Zeitschlitzes einfach in Ruhe. Bei den *Sporadic Frames* ist es möglich und üblich, für mehrere Botschaften mit unterschiedlichen Identifiern denselben Zeitschlitz zu reservieren, wobei bei der Konfiguration eine statische Priorität für die Botschaft vergeben wird, so dass der Scheduler im Master jeweils nur die höchstpriore Botschaft versendet.

*Event Triggered Frames* werden verwendet, um mehrere Slave-Steuergeräte mit einer einzigen Botschaft zyklisch abzufragen. *Event Triggered Frames* sind der einzige Botschaftstyp, bei dem mehrere Slave-Funktionen so konfiguriert werden, dass sie auf dieselbe Botschaft antworten können. Tatsächlich antwortet ein Slave allerdings nur dann, wenn sich die zu dieser Botschaft gehörenden Daten innerhalb des Steuergerätes geändert haben. *Event Triggered Frames* sind daher nur für solche Daten geeignet, die zwar von mehreren Geräten geliefert werden, bei denen sich der Wert in der Regel aber nur in einem Gerät tatsächlich ändert. Sinnvoll wäre dies beispielsweise bei der Abfrage von Türkontaktschaltern, wenn zu erwarten ist, dass nicht mehrere Türen gleichzeitig geöffnet oder geschlossen werden. Falls sich die zugehörigen Daten in keinem der Slave-Steuergeräte geändert haben, antwortet keines der Steuergeräte. Haben sich ausnahmsweise die Daten in mehreren Slave-Steuergeräten gleichzeitig geändert und versuchen daher mehrere Geräte, auf die Header-Botschaft zu antworten, so kommt es bei der Antwort zu einer Kollision.

Die Slaves erkennen die Kollision durch Rücklesen der gesendeten Daten und brechen das Senden vorzeitig ab. Der Master erkennt dies als Empfangs- oder Timeout-Fehler. In den nächsten diesem *Event Triggered Frame* zugeordneten Zeitschlitzen fragt der Master die Slave-Steuergeräte der Reihe nach mit normalen Botschaften (d. h. mit separaten Identifiern) ab, bevor er wieder den *Event Triggered Frame* sendet. D. h. für jedes Steuergerät, das auf den Identifier eines bestimmten *Event Triggered Frame* reagiert, muss es zusätzlich auch noch einen jeweils eindeutigen *normalen* Identifier geben. Aufgrund der Verzögerungen bei Kollisionen sind *Event Triggered Frames* nicht mehr streng deterministisch. Bei LIN V2.1 wird die Kollisionsbehandlung präziser spezifiziert und gefordert, dass der Master nach einer erkannten Kollision auf eine andere Botschaftstabelle umschalten soll, die Kollisionsfreiheit gewährleistet. Diese soll einmalig abgearbeitet werden, bevor wieder auf die ursprüngliche Botschaftstabelle zurückgeschaltet wird.

### 3.2.4 LIN Transportschicht und ISO Diagnose über LIN

Da Steuergeräte am LIN-Bus wie alle übrigen Kfz-Steuergeräte für die Werkstattdiagnose zugänglich sein müssen, implementiert in der Regel das Master-Steuergerät eine der Standard-Diagnoseschnittstellen (KWP 2000 mit CAN bzw. UDS mit CAN, siehe Kap. 5). Der Datenaustausch mit den LIN-Slave-Steuergeräten dagegen erfolgte in LIN V1.3 ausschließlich über normale LIN-Botschaften mit den üblichen LIN-Botschafts-Identifiern, so dass die LIN-Slaves selbst keine KWP 2000- oder UDS-Protokollsoftware enthalten müssen. Mit LIN V2.0 dagegen wurde die optionale Möglichkeit eingeführt, das KWP 2000- bzw. UDS-Protokoll nach ISO 15765 und ISO 14229 mit Hilfe eines Transportprotokolls ähnlich ISO TP (siehe Kap. 4) über den LIN-Bus zu „tunneln" (*LIN Transport Layer*). In diesem Fall muss das LIN-Slave-Steuergerät zusätzlich zum LIN-Protokoll die Anwendungsschicht des UDS bzw. KWP 2000-Protokolls realisieren, was dem ursprünglichen LIN-Ziel, möglichst einfache Slave-Steuergeräte zu ermöglichen, allerdings nicht unbedingt entspricht. Mit LIN V2.1 wurde die pauschale Forderung, das Diagnoseprotokoll zu unterstützen, abschwächend spezifiziert, indem sogenannte Geräteklassen (*Diagnostic Class*) eingeführt wurden. Bei Slave-Geräten der Klasse I, einfachen Sensor-Aktor-Geräten, wird lediglich die Unterstützung von Single Frame-Botschaften für die dynamische Gerätekonfiguration gefordert (siehe Abschn. 3.2.6). Geräte der Klasse II sollen alle Transportprotokollbotschaften (siehe unten) unterstützen, müssen aber nur den UDS/KWP 2000-Diagnosedienst *Read Data By Identifier* implementieren, um Steuergerätedaten, z. B. die Gerätekennung, auslesen zu können. In der höchsten Klasse III wird zusätzlich die Unterstützung von Diagnosesitzungen (*Diagnostic Session Control*), Lesen und Ansteuern von Steuergeräteein- und -ausgängen (*Input Output Control*), des Fehlerspeichers (*Read/Clear Diagnostic Trouble Code Information*) sowie optional der Funktionen für die Flash-Programmierung (siehe Tab. 5.19 und 5.20) gefordert.

Beim Tunneln des UDS/KWP 2000-Protokolls über LIN werden UDS/KWP 2000-Diagnose-Request-Botschaften vom Master-Steuergerät/Gateway (MRF) als LIN-Bot-

**Tab. 3.7** Geräteklassen für Transportprotokoll und Diagnose

| Geräteklasse | Transportschicht nach ISO 15765-2 (Kap. 4) | Diagnostic Service Identifier SID nach ISO 15765-3 bzw. 14229 (siehe Kap. 5) |
|---|---|---|
| I | Nur Single Frame | Konfigurationsdienste siehe Abschn. 3.2.6 (B2h, B7h, optional B0h, B3h … B6h) |
| II | Single, First und Consecutive Frame | Konfigurationsdienste wie Klasse I Diagnosedienste 22h, 2Eh |
| III | | Wie Klasse II, zusätzlich Diagnosedienste 10h, 14h, 19h, 2Fh, optional 31h und Flash-Programmierung |

schaften mit dem Identifier 3Ch versendet und Diagnose-Response-Botschaften der Slaves (SRF) mit dem Identifier 3Dh abgefragt. Das ISO 15765-2-Botschaftsformat mit Single Frames sowie das Multi-Frame-Format mit First Frames und Consecutive Frames nach Abb. 4.2 wird in die 8 Datenbytes der LIN-Botschaft abgebildet, die bei CAN notwendige Flusssteuerung mit Flow-Control-Botschaften ist bei LIN wegen des zeitsynchronen Botschaftsverkehrs nicht notwendig.

Auf LIN-Ebene sind diese Botschaften als Broadcast-Botschaften zu verstehen, d. h. alle LIN-Slave-Steuergeräte, welche die LIN-ISO-Diagnose unterstützen, müssen die Request-Botschaften mit dem Identifier 3Ch empfangen. Die Auswahl des Steuergerätes, für das die Diagnoseanfrage bestimmt ist, erfolgt über die ISO-Adresse (siehe Abschn. 5.1.3), bei LIN als *Node Diagnostic Address NAD* bezeichnet, die im ersten Datenbyte übertragen wird. Als Geräteadressen sind die Werte 1 bis 7Dh vorgesehen. Mit 7Eh ist eine funktionale Adressierung möglich und 7Fh dient als Broadcast Adresse. Der Slave verwendet in seiner Antwort seine eigene Adresse. Die Master-Request- und Slave-Response-Botschaften mit den Identifiern 3Ch und 3Dh müssen immer 8 Datenbytes lang sein, *leere* Datenbytes werden mit FFh aufgefüllt.

Auch für Diagnosebotschaften muss ein Zeitschlitz in der Botschaftstabelle festgelegt werden, der aber nur verwendet wird, wenn das Master-Steuergerät eine Request-Botschaft abzusetzen hat oder eine Slave-Response-Botschaft erwartet.

Während ISO 15765-2, KWP 2000 und UDS relativ detailliert enge Zeitschranken für die Überwachung des Botschaftsverkehrs vorschreiben (siehe z. B. Tab. 3.7), enthält LIN V2.0 keine konkreten Vorgaben. Seit LIN V2.1 wird immerhin gefordert, dass die Zeitdauer zwischen Sendeanforderung und tatsächlichem Versenden einer Botschaft sowie zwischen zwei aufeinanderfolgenden Consecutive Frames nicht mehr als eine Sekunde betragen soll, wobei in einer konkreten Anwendung auch kürzere Zeiten konfiguriert werden dürfen.

### 3.2.5 LIN Configuration Language

Die LIN-Spezifikation sieht eine Konfigurationssprache vor, mit deren Hilfe die Netzwerkteilnehmer (*Nodes*), die zu übertragenden Daten (*Signals*) und die daraus zusammenge-

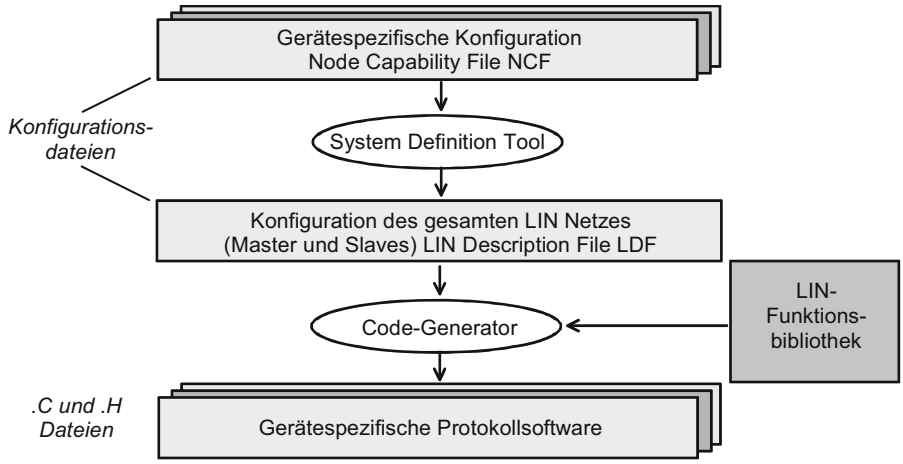


**Abb. 3.14** LIN-Konfiguration

stellten Botschaften (*Frames*) spezifiziert werden können. Obwohl es sich bei den Konfigurationsdateien um einfache Textdateien handelt, die theoretisch mit jedem Texteditor erstellt werden können, verwendet man in der Praxis spezielle Konfigurationswerkzeuge. Aus den Konfigurationsdateien für die einzelnen Netzwerkteilnehmer (*Node Capability File NCF*) lässt sich dann die Konfigurationsdatei für das Gesamtnetz (*LIN Description File LDF*) erzeugen (Abb. 3.14). Daraus kann dann ein weiteres Werkzeug automatisch einen Satz von C-Code- und Header-Dateien für die Steuergeräte generieren, die die Master- und Slave-Funktionen implementieren.

Außerdem ist es möglich, auf Basis der LIN-Konfigurationsdatei das Netzwerkverhalten zu simulieren.

Der frei verfügbare LIN-Standard definiert allerdings nur die Konfigurationssprache sowie die Programmierschnittstelle (API) zwischen der LIN-Protokoll-Software und der Anwendungs- bzw. UART-Treibersoftware des Steuergerätes (Abb. 3.15). Bei den Werkzeugen dagegen handelt es sich um kommerzielle Produkte, die von verschiedenen Herstellern angeboten werden.

Die LIN-Konfigurationsdatei enthält folgende Hauptabschnitte (Tab. 3.8):

- Kopf mit Versionsnummer des Busprotokolls und der Konfigurationssprache sowie Bitrate des Netzes.
- `Nodes{...}`: Definiert symbolische Namen für das Master-Steuergerät und für alle Slave-Steuergeräte. Weiterhin werden für das Master-Steuergerät die Scheduling-Periodendauer und deren Zeittoleranz (*Jitter*) angegeben (siehe unten). Im Abschnitt `Node_attributes{...}`, der in LIN V2.0 neu ist, werden für jedes Steuergerät die unterstützte Protokollversion, die Diagnoseadresse NAD, die *LIN Product Identification*

**Tab. 3.8** LIN-Konfigurationsdatei

```
LIN_description_file;
LIN_protocol_version = "1.3";
LIN_language_version = "1.3";
LIN_speed = 19.2 kbps;

Nodes {
       Master:CEM,5 ms, 0.1 ms; // Master-Steuergerät, Scheduling-Periode 5ms
       Slaves:LSM,CPM;          // Slave-Steuergeräte
}
Signals {                       // Definition von Signalen
       CPMOutputs:10,0,CPM,CEM;
       HeaterStatus:4,0,CPM,CEM;
       CPMGlowPlug:7,0,CPM,CEM;
       WaterTempLow:8,0,CPM,CEM,LSM;
       WaterTempHigh:8,0,CPM,CEM,LSM;
       CPMFuelPump:7,0,CPM,CEM;
       . . .
}
Signal_encoding_types {
       Temp {
              physical_value,0,250,0.5,-40,"degree";
              physical_value,251,253,1,0,"undefined";
              logical_value,254,"out of range";
              logical_value,255,"error";
       }
       . . .
}
Signal_representations {
       Temp:WaterTempLow,WaterTempHigh;
       . . .
}
Frames {                        // Definition von Botschaften
       VL1_CEM_Frm1:32,CEM,3 {  // - - - Botschaft 1
              . . .
       }
       VL1_CPM_Frm1:50,CPM {    // - - - Botschaft 2
              CPMOutputs,0;
              HeaterStatus,10;
              WaterTempLow,32;
              WaterTempHigh,40;
              CPMFuelPump,56;
       }
       . . .
}
Schedule_tables {               // Botschaftstabelle
       VL1_ST1 {
              VL1_CEM_Frm1 delay 15 ms;   // - - - Botschaft 1 alle 15ms
              VL1_CPM_Frm1 delay 20 ms;   // - - - Botschaft 2 alle 20ms
              . . .
       }
}
```

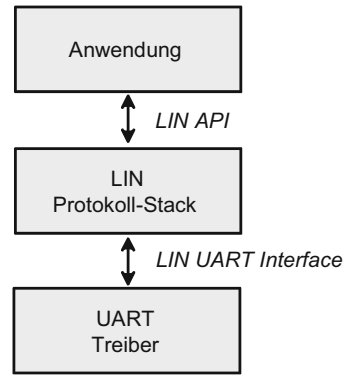


**Abb. 3.15** Schnittstellen der LIN-Protokoll-Software (LIN Protocol Stack)

(siehe Abschn. 3.2.6) und einige weitere Parameter angegeben, z. B. welche Botschaften dynamisch konfigurierbare LIN-Identifier haben.

- `Signals{...}`: Definiert symbolische Namen für alle zu übertragenen Datenwerte, deren Größe (1…16 bit bzw. 1 bis 8 Byte) sowie einen Initialisierungswert, der verwendet wird, wenn noch kein aktueller Wert von der Applikation vorgegeben bzw. vom Bus empfangen wurde. Außerdem wird für jedes Signal angegeben, welches Steuergerät (*Node*) das Signal senden (*publish*) kann und welche Steuergeräte das Signal empfangen (*subscribe*) wollen. Um kurze Botschaften zu erhalten, können mehrere Signale, die kürzer sind als 16 bit, bei der Übertragung zu einem oder zwei Datenbytes zusammengefasst werden. Zusammengehörige Datenstrukturen, die mehr als 1 Byte benötigen, können zu einem *Byte Array* zusammengefasst werden (in V1.3 noch als *Signal Group* bezeichnet).
- `Signal_encoding_types{...}`: Optional. Definiert den Wertebereich der übertragenen Hexadezimalwerte sowie die Umrechnung in den echten physikalischen Wert nach dem Umrechnungsschema

$$\text{Physikal. Wert} = \text{Skalierungsfaktor} \cdot \text{Hexadezimalwert} + \text{Offset}.$$

- Für jede Umrechnungsformel wird ein symbolischer Name definiert. Im Abschnitt `Signal_representations{...}` kann einem Signal dann eine Umrechnungsformel zugeordnet werden.
- `Frame{...}`: Definiert für jede Botschaft einen symbolischen Namen, den zugehörigen Identifier, das sendende Steuergerät und legt fest, aus welchen Signalen die Botschaft aufgebaut ist. Für jedes Signal wird die Position innerhalb der Botschaft (über einen Bitoffset beginnend ab dem ersten Datenbit der Botschaft) festgelegt. Signale mit weniger als 8 bit können innerhalb eines Bytes zusammengefasst werden.
- `Schedule_tables{...}`: Definiert eine Tabelle mit allen Botschaften (*Frames*), die übertragen werden sollen, samt den zugehörigen Wiederholperioden. Aus dieser Tabelle entnimmt der Master, wann welche Header-Botschaft zu senden ist. Bei der Erzeugung der Konfigurationsdateien muss durch den Anwender bzw. das Konfigurationswerkzeug sichergestellt werden, dass die Wiederholperioden der einzelnen Bot-

schaften nur ganzzahlige Vielfache der Scheduling-Periodendauer sind und dass die Wiederholperioden so groß gewählt werden, dass die Übertragung aller in der Botschaftstabelle definierten Botschaften unter Berücksichtigung der Übertragungsdauer und des Jitters der Scheduling-Periode in jedem Fall tatsächlich möglich ist. Mit LIN V2.0 wurden einige zusätzliche Attribute für diesen Konfigurationsabschnitt eingeführt. Üblich sind mehrere Schedule-Tabellen für unterschiedliche Betriebszustände des Systems.

Die folgenden Abschnitte sind erst seit LIN V2.0 vorhanden:

- Unter `Diagnostic_frames{...}` können vollständige Diagnosebotschaften (LIN-Identifier 3Ch und 3Dh) und mit `Diagnostic_signals{...}` die darin enthaltenen Datenwerte definiert werden.
- Unter `Sporadic_Frames{...}` und `Event_triggered_frames{...}` werden die in Abschn. 3.2.6 beschriebenen *dynamischen Botschaften* konfiguriert.

*Node Capability Files*, die die Fähigkeiten einzelner Geräte beschreiben, verwenden nahezu dieselbe Syntax, wobei sich aber nur ein Teil der o. g. Angaben findet und bei einigen Daten, z. B. den Wiederholperioden einer Botschaft, der zulässige Wertebereich statt eines einzelnen Wertes angegeben werden kann.

Bei LIN V2.1 wurden einige Details verändert, um die neu hinzugekommenen Botschaften spezifizieren zu können. Alternativ zu LDF/NCF kann auch das neuere FIBEX-Konfigurationsformat (siehe Abschn. 6.3) eingesetzt werden, das sich neben LIN auch für CAN, FlexRay und andere Busse eignet.

### 3.2.6 Dynamische Konfiguration von LIN-Slave-Steuergeräten

Mit LIN V2.0 wurde ein Mechanismus neu eingeführt, um LIN Slave-Steuergeräte dynamisch zu konfigurieren. Dahinter steckt der Gedanke, dass einfache Steuergeräte vom Hersteller mit einer festen Grundkonfiguration ausgeliefert und dann erst im eingebauten Zustand durch das Master-Steuergerät systemspezifisch konfiguriert werden. Üblich ist dies z. B. bei Sitz- oder Lüftersteuergeräten. Damit entfällt die Lagerhaltung unterschiedlicher Varianten von Slave-Steuergeräten. In Anlehnung an die PC-Welt wird die dynamische Konfiguration auch als *LIN Plug-and-Play* bezeichnet.

In der Grundkonfiguration sind für die Steuergeräte-Diagnoseadresse NAD und alle Botschaften vordefinierte LIN Identifier festgelegt, die dann während des Konfigurationsprozesses durch das Master-Steuergerät umkonfiguriert werden. Im Hinblick auf die schnelle Betriebsfähigkeit des Bussystems nach Einschalten der Spannungsversorgung wird man den Konfigurationsvorgang möglichst nur in der Fahrzeugfertigung oder beim Gerätetausch in der Werkstatt durchführen, so dass LIN V2.0-Steuergeräte sinnvollerweise einen Flash-ROM- oder EEPROM-Speicher für die dynamischen Konfigurationsdaten besitzen sollten.

Das Steuergerät enthält eine feste Gerätekennzeichnung, die *LIN Product Identification*, die in einem 16 bit Wert, der vom LIN-Konsortium vergeben wird, eindeutig den Hersteller des Gerätes und in einem vom Hersteller festgelegten 16 bit-Wert die Funktion und in einem weiteren 8 bit-Wert eine Versionsnummer des Steuergerätes angibt.

Für die Konfiguration wird das bereits für die Diagnosebotschaften definierte Botschaftsformat nach Abb. 3.13 mit den LIN-Identifiern 3Ch für die Botschaften vom Master-Steuergerät und 3Dh für die Antworten des Slave-Steuergerätes verwendet. Als Service Identifier SID werden die bei der ISO-Diagnose als herstellerspezifische deklarierten Werte SID=B0h … B4h eingesetzt, ab LIN V2.1 zusätzlich B5h … B7h. Diese Botschaften werden, wie in Abschn. 3.2.4 dargestellt, als Broadcast-Botschaften versendet. Die Auswahl des angesprochenen Steuergerätes erfolgt über das NAD ISO Adressbyte (1. Datenbyte der Botschaft). Zulässige Werte sind 1 … 126. Mit LIN V2.1 wird NAD = 7Eh = 126 für die funktionale Adressierung von LIN-Steuergeräten bei der Diagnose nach ISO reserviert (siehe Abschn. 5.1.3).

Mit Hilfe einer Diagnosebotschaft mit SID = B1h (*Assign Frame Identifier*) kann das Master-Steuergerät einem im Slave-Steuergerät vordefinierten LIN-Botschafts-Identifier einen neuen Wert zuweisen. Neben dem alten und dem neuen Identifier muss der Master dabei in dieser Botschaft auch noch die Herstellerkennung mitsenden, um die Wahrscheinlichkeit zu erhöhen, dass keine Fehlkonfiguration erfolgt. Ab LIN V2.1 soll diese Botschaft durch eine neue Botschaft mit SID = B7h (*Assign Frame Identifier Range*) ersetzt werden, die inhaltlich dasselbe leistet, aber mehrere Identifier gleichzeitig festlegen kann.

Zuvor oder anschließend zur Überprüfung kann der Master mit Hilfe einer Diagnosebotschaft mit SID = B2h (*Read by Identifier*) die aktuelle Konfiguration auslesen. Neben dem Auswahlbyte, das den zu lesenden Wert auswählt, muss der Master dabei aus Sicherheitsgründen die Herstellerkennung und die Funktionskennung mitsenden. Sendet er als Auswahlbyte den Wert 0, erhält er als Antwort die vollständige *LIN Product Identification*, beim Wert 1 die Seriennummer des Geräteherstellers und mit allen anderen Werten im Bereich 16 … 63 die LIN-Botschaftskennungen der im Gerät vorkonfigurierten Botschaften. Mit LIN V2.1 wurde der Bereich auf 32 … 63 eingeschränkt.

Optional ist die Möglichkeit, mit SID = B0h (*Assign NAD*) bzw. SID = B3h (*Conditional Change NAD*) die Steuergeräte-Diagnoseadresse neu festzulegen. Dies ist notwendig, wenn in der Default-Konfiguration mehrere Steuergeräte dieselbe NAD verwenden. Neben der neuen Adresse wird aus Sicherheitsgründen wieder die Hersteller- und Funktionskennung mitgesendet und vom Slave-Steuergerät geprüft, bevor der neue Wert akzeptiert wird. Dieser Mechanismus versagt allerdings, wenn mehrere exakt baugleiche Geräte desselben Herstellers und derselben Default-NAD-Adresse verbaut werden, da diese dann alle auf die Diagnosebotschaft antworten und es zu Kollisionen kommt. Die LIN V2.0-Spezifikation schlägt in diesem Fall vor, dass das Slave-Steuergerät seinen Default-NAD-Wert selbstständig z. B. auf Basis einer eventuell vorhandenen eindeutigen Seriennummer verändert, ohne dies zu konkretisieren. In diesem Fall muss das Master-Steuergerät bei der Konfiguration verschiedene NAD-Adressen „durchprobieren", bis es von den Slave-Steuergeräten korrekte Antworten ohne Kollisionen erhält. LIN V2.1 verweist hierzu auf ein separates Dokument

für ein noch zu definierendes *Slave Node Position Detection* (SNPD) Verfahren und definiert dafür den neuen Dienst SID = B5h (*Assign NAD by SNPD*).

Ebenfalls neu ist in LIN V2.1 die optionale *Save Configuration* Botschaft (SID = B6h), mit der ein Slave-Steuergerät aufgefordert wird, die dynamisch erzeugte Konfiguration dauerhaft abzuspeichern.

Für den Gerätehersteller ist vorgesehen, mit SID = B4h (*Data Dump*) herstellerspezifische Daten mit dem Steuergerät auszutauschen. Dieser Mechanismus ist für die Fertigung und den Test beim Gerätehersteller gedacht und soll nicht im Systemverbund im Fahrzeug verwendet werden.

### 3.2.7 LIN Application Programming Interface (API)

Für die Entwickler von LIN-Protokoll-Software enthält die LIN-Spezifikation einen Vorschlag für eine einheitliche Programmierschnittstelle (API), was insbesondere dann sinnvoll ist, wenn die LIN-Protokoll-Stack-Software durch einen automatischen Codegenerator erzeugt oder von einem Softwarelieferanten zugeliefert wird (Abb. 3.15).

Das API enthält folgende wesentlichen Funktionsgruppen:

- **Initialisieren der LIN-Protokoll-Software** (`l_sys_init()`), Konfigurieren (Setzen des Baudratenregisters usw. im UART mit `l_ifc_init()` bzw. `l_ifc_ioctl()`). Bei LIN 1.3 und 2.0 gab es noch Funktionen zum expliziten Aktivieren und Deaktivieren der LIN-Schnittstelle (Starten und Beenden des Empfangens und Sendens von Botschaften mit `l_ifc_connect()` und `l_ifc_disconnect()`). Mit `l_ifc_goto_sleep()` kann die Applikation im Master-Steuergerät das Aussenden der Sleep-Botschaft veranlassen, die die LIN-Aktivitäten aller Steuergeräte abschaltet, bis eine Applikation in einem der Steuergeräte mit `l_ifc_wake_up()` den Bus wieder *aufweckt*.
- **Lesen und Schreiben** (`l_..._rd()` bzw. `l_..._wr()`) von 1 bit, 8 bit oder 16 bit-Daten bzw. von Byte Arrays (im LIN-Sprachgebrauch: Signale). Beim Lesen wird der jeweils letzte empfangene Wert gelesen, beim Schreiben wird der neue Wert in eine Signal-Tabelle eingetragen. Das eigentliche Senden erfolgt erst, wenn das Steuergerät eine Header-Botschaft mit dem entsprechenden Identifier erhält und das Gerät mit der Daten-Antwortbotschaft reagiert.
- Der **innere Zustand der LIN-Protokoll-Software** wird in verschiedenen implementierungsabhängigen Statusworten, so genannten Flags, gespeichert, die von der Anwendungssoftware gelesen und gelöscht werden können (`l_flg_tst()`, `l_flg_clr()`). Damit kann z. B. abgefragt werden, ob ein bestimmtes Signal empfangen wurde. Mit LIN V2.0 wurde der `l_ifc_read_status()` Aufruf eingeführt, mit dem die Anwendung abfragen kann, welcher Identifier als letztes empfangen wurde und ob Sende- bzw. Empfangsfehler aufgetreten sind.

- Als **Interface zum UART-Treiber** dienen `l_ifc_rx()` und `l_ifc_tx()`. Diese Funktionen werden aufgerufen, wenn ein Zeichen empfangen oder gesendet wurde. Falls der UART bzw. sein Softwaretreiber selbstständig in der Lage ist, den *Sync Break* zu erkennen, wird die Funktion `l_ifc_aux()` aufgerufen.
- Manche LIN-Protokollstapel müssen für interne Verwaltungsoperationen kurzzeitig die **Interrupts des Steuergerätes sperren** können. Dazu muss die Anwendungssoftware die Funktionen `l_sys_irq_disable()` und `l_sys_irq_restore()` zur Verfügung stellen.
- Für die **Master-Funktion** sind `l_sch_tick()` und `l_sch_set()` vorgesehen. Mit `l_sch_set()` wird eine Botschaftstabelle (Schedule Table) aktiviert, d. h. eine Tabelle, die die Liste der periodisch zu übertragenden Botschaften enthält. Der Scheduler arbeitet in einem festen Zeitraster, der Scheduler-Periode. Der Aufruf von `l_sch_tick()` zeigt der Protokollsoftware die nächste Periode an.
- Für die **Transportschicht** und die **ISO-Diagnose über LIN** schlägt die LIN-Spezifikation V2.0 zwei verschiedene APIs vor. Das Master-Steuergerät, das als Gateway zwischen dem CAN-Bus und dem LIN-Bus fungiert, muss im Wesentlichen das Kopieren der Botschaften durchführen, ohne deren Inhalt zu interpretieren. Die dafür vorgesehenen *Raw-API*-Funktionen `ld_put_raw()` und `ld_get_raw()` tragen eine ISO-Diagnosebotschaft in den LIN-Botschaftsspeicher ein bzw. lesen sie aus. Da sich die Übertragungsgeschwindigkeiten von CAN und LIN deutlich unterscheiden, sind im CAN-LIN-Gateway üblicherweise Puffer, z. B. FIFOs, vorhanden. Mit `ld_raw_tx_status()` bzw. `ld_raw_rx_status()` kann der Zustand der Puffer abgefragt werden. Für LIN-Slave-Steuergeräte sind die *Cooked-API*-Funktionen sinnvoller. Hierbei werden mit `ld_receive_message()` und `ld_send_message()` vollständige ISO-Botschaften (ab dem *Diagnostic Service Identifier SID* mit bis zu 4095 Bytes) gelesen und geschrieben. Das Segmentieren und Desegmentieren findet innerhalb der LIN-Protokollsoftware statt. Die Funktionen arbeiten asynchron, d. h. sie kehren sofort zurück, bevor die Daten tatsächlich empfangen bzw. gesendet wurden. Mit `ld_tx_status()` und `ld_rx_status()` kann die Anwendung herausfinden, ob der Vorgang erfolgreich abgeschlossen wurde.
- Für die mit LIN V2.0 neu eingeführte **dynamische Konfiguration** sind für das Master-Steuergerät die Funktionen `ld_assign_frame_id()`, `ld_read_by_id()`, `ld_assign_NAD()` und `ld_conditional_assign_NAD()` vorgesehen, die die in Abschn. 3.2.6 beschriebenen Botschaften erzeugen. Der Erfolg bzw. Misserfolg einer Konfigurationsbotschaft kann mit `ld_check_response()` und `ld_is_ready()` abgefragt werden. Mit `ld_save_configuration()` fordert der Master ein oder mehrere Slave-Steuergeräte auf, die Konfiguration zu übernehmen, die dann dort mit `ld_set_configuration()` gespeichert bzw. mit `ld_read_configuration()` wieder gelesen werden kann.

**Tab. 3.9** Botschaftslängen für $f_{\text{Bit}} = 1 / T_{\text{bit}} = 19{,}2\,\text{kbit/s}$

| $n_{\text{Data}}$ | $T_{\text{Header, min}}$ | $T_{\text{Response, min}}$ | $T_{\text{Frame, min}}$ | $T_{\text{Frame, max}}$ |
|---|---|---|---|---|
| 1 byte | 1,8 ms | 1,0 ms | 2,8 ms | 3,9 ms |
| 8 byte | | 4,7 ms | 6,5 ms | 9,0 ms |

### 3.2.8 Zeitverhalten von LIN-Systemen

Die Länge einer LIN Botschaft (Tab. 3.9) ist

$$T_{\text{Frame}} = T_{\text{Header}} + T_{\text{Response}} \tag{3.12}$$

mit $T_{\text{Header,min}} = 34\,\text{bit} \cdot T_{\text{bit}}$ und $T_{\text{Response,min}} = \frac{10}{8}(n_{\text{Data}} + 8\,\text{bit}) \cdot T_{\text{bit}}$.

Dabei ist $n_{\text{Data}} = 0, \ldots, 64\,\text{bit}$ die Anzahl der Datenbits, die in Stufen von 8 bit variabel ist. Der Faktor 10 / 8 berücksichtigt, dass UART-üblich je Byte zusätzlich ein Start- und ein Stopbit eingefügt werden. Die minimale Länge ergibt sich, wenn das *Sync Break Feld* die kleinste zulässige Länge (13 bit) hat und keine Pausen zwischen den einzelnen Bytes bzw. zwischen Header und Response auftreten. Als maximal zulässige Länge gibt die LIN-Spezifikation vor

$$T_{\text{Frame,max}} = 1{,}4 \cdot \left(T_{\text{Header,min}} + T_{\text{Response,min}}\right). \tag{3.13}$$

Um die Implementierung der LIN-Treibersoftware zu vereinfachen, werden LIN-Botschaften üblicherweise zeitsynchron gesendet, wobei das Master-Steuergerät durch das Senden der Header-Botschaften das Zeitraster (*Frame Slots*) vorgibt und über den Identifier im Header bestimmt, welche Botschaft in diesem Zeitschlitz gesendet wird. Die Reihenfolge und Wiederholperiode aller Botschaften wird statisch während der Entwicklungsphase des Netzes festgelegt und im Master-Steuergerät in einer so genannten Botschaftstabelle (*Scheduling Table*) abgelegt, die von der Master-Funktion abgearbeitet wird (*Message Scheduling*). Dabei können mehrere solche Botschaftstabellen vorhanden sein, zwischen denen die Applikationssoftware im Master-Steuergerät umschalten kann. Bei der Konfiguration der Botschaftstabellen muss der Entwickler, in der Regel mit Hilfe eines Konfigurationswerkzeuges, sicherstellen, dass alle Botschaften mit den festgelegten Wiederholperioden stets übertragen werden können. Dadurch ist das Übertragungsverhalten des LIN-Busses deterministisch.

Die Dauer eines Zeitschlitzes $T_{\text{Slot}}$ im Zeitraster (*Scheduling Periode*) wird so gewählt, dass sie unter Berücksichtigung aller Toleranzen (*Scheduling Jitter*) stets größer ist als die Übertragungsdauer $T_{\text{Frame}}$ der Botschaft, d. h.

$$T_{\text{Slot}} > T_{\text{Frame,max}}. \tag{3.14}$$

Für ein LIN-Bussystem mit einer Bitrate von 19,2 kbit/s etwa wäre $T_{\text{Slot}} = 10\,\text{ms}$ sinnvoll. Nach jeweils $N$ Zeitschlitzen (*Slots*) wiederholt sich die Botschaftssequenz (Abb. 3.16), so

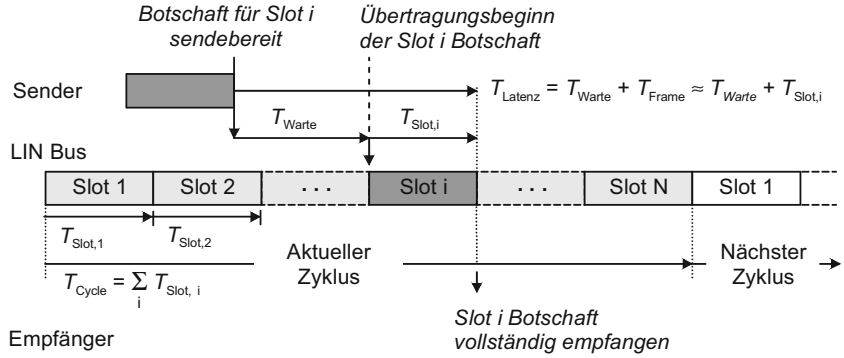


**Abb. 3.16** LIN Botschaftszyklus

dass die Zykluszeit

$$T_{\text{Zyklus}} = \sum_{=1}^{N} T_{\text{Slot},i} \tag{3.15}$$

ist. Bezogen auf eine einzelne Botschaft sind auch höhere Wiederholraten möglich, indem für eine Botschaft mehrere Zeitschlitze innerhalb eines Zyklus eingeplant werden. Die Latenzzeit bei der Übertragung setzt sich wie bei CAN aus der Zeit $T_{\text{Warte}}$ bis zum Sendebeginn und der eigentlichen Übertragungsdauer $T_{\text{Frame}}$ zusammen:

$$T_{\text{Latenz}} = T_{\text{Warte}} + T_{\text{Frame}}. \tag{3.16}$$

Im günstigsten Fall wird die Botschaft auf der Sendeseite direkt vor Beginn des zugeordneten Zeitschlitzes zum Senden bereitgestellt, so dass die Wartezeit entfällt. Im ungünstigsten Fall stehen die Sendedaten erst nach Beginn des Zeitschlitzes zur Verfügung und können daher erst einen vollen Zyklus später gesendet werden:

$$T_{\text{Latenz,min}} = T_{\text{Frame}} < T_{\text{Latenz}} < T_{\text{Latenz,max}} = T_{\text{Zyklus}} + T_{\text{Frame}}. \tag{3.17}$$

Um den Jitter möglichst gering zu halten, sollte die Anwendungssoftware, die die Daten bereitstellt, daher mit dem Zyklus des LIN Bussystems synchronisiert werden.

### 3.2.9 Zusammenfassung LIN – Layer 1 und 2

- Zeichenorientiertes UART-basiertes Übertragungsprotokoll mit bidirektionaler Ein-Draht-Leitung (wie K-Line), Bitrate zwischen 1 … 20 kbit/s (üblich 19,2 kbit/s), Signalpegel $U_B$ (Batteriespannung), entwickelt mit der Zielsetzung eines Low-Cost-Bussystems für einfache Sensor-Aktor-Anwendungen, maximale Nutzdatenrate unter 1,2 KB/s (bei 20 kbit/s).

- Anzahl der Busteilnehmer aus elektrischen Gründen und wegen der begrenzten Anzahl von Botschafts-Identifiern typ. < 16, relative kurze Buslängen < 40 m.
- Ein Steuergerät als *Master* sendet periodisch Botschafts-Header, eines der Geräte (*Slaves*) antwortet mit max. 8 Datenbytes und einem Prüfsummenbyte.
- Senden aller Botschaften periodisch in festem Zeitraster. Senderaster im Master als Botschaftstabelle (Schedule Table) statisch konfiguriert, unterschiedliche Botschaftstabellen für verschiedene Betriebszustände möglich. Typ. Sendezeitraster 10 ms bei einer max. Botschaftsdauer von ca. 9 ms (Botschaft mit 8 Byte Nutzdaten und Bittakt 20 kbit/s), intern arbeitet der Master dann z. B. mit 5 ms.
- Inhalt der Botschaft wird durch *Identifier* (im vom Master gesendeten Header) ausgewählt (inhaltsbezogene Adressierung wie bei CAN). 60 verschiedene Botschaften möglich. Je zwei Identifier sind für spezielle Botschaften und für zukünftige Protokollerweiterungen vorgesehen.
- Anforderungen an die Bittakt-Genauigkeit der Slaves und gesamtes Protokoll-Timing gering, d. h. Slaves können ohne eigenen Quarz implementiert werden.
- Primitive Fehlerüberwachung (Rücklesen der Bussignale, Prüfsumme, Antwortzeitüberwachung), keine Fehlerkorrektur. Fehlerbehandlung auf Anwendungsebene hersteller- und anwendungsabhängig.
- Master-Steuergerät häufig als Gateway zu anderen Bus-Systemen, z. B. CAN.
- Mit LIN V2.x optionale Botschaften (*Transportprotokoll*), mit denen größere Datenblöcke oder ISO-Diagnosebotschaften über LIN versendet werden können.

## 3.3 FlexRay

FlexRay wurde vor dem Hintergrund zukünftiger X-by-Wire-Anwendungen (X = Brake, Steer, …) entwickelt [11–13] und ist heute als ISO 17458 standardisiert. Obwohl dieses Anforderungsprofil in weiten Teilen dem von CAN gleicht, erschien vielen Fachleuten eine Neuentwicklung notwendig und sinnvoll. Haupttriebfeder war zunächst, dass die bei CAN sinnvoll erreichbare Bit- und damit Datenrate aufgrund des CAN-Grundprinzips der bitweisen Arbitrierung auf maximal 1 Mbit/s und die Bus-Topologie auf ein reines Linienbussystem mit (bei hohen Bitraten) kurzen Stichleitungen beschränkt ist. Dies zwingt teilweise zu einer aus fertigungstechnischer Sicht suboptimalen Kabelführung im Fahrzeug. Dazu kommt, dass CAN zwar eine hohe Übertragungssicherheit gewährleistet, aber Schwächen bei extremen Sicherheitsanforderungen hat. Schwer wiegt dabei, dass CAN zunächst ein rein einkanaliges System ist, das bei Ausfall der Busverbindung versagt. Zweikanalige Aufbauten sind natürlich möglich, aber Mechanismen zur Synchronisation und Plausibilitätsprüfung zwischen den Kanälen fehlen und müssen daher mühsam softwaremäßig nachgebildet werden. Zum anderen ist der asynchrone Buszugriff bei CAN nicht streng deterministisch. Dies führt dazu, dass nur für die höchstpriore Botschaft eine maximale Latenzzeit bei der Übertragung garantiert werden kann. Für weniger hoch priorisierte Botschaften lässt sich unter gewissen Voraussetzungen (ähnlich wie beim Task-

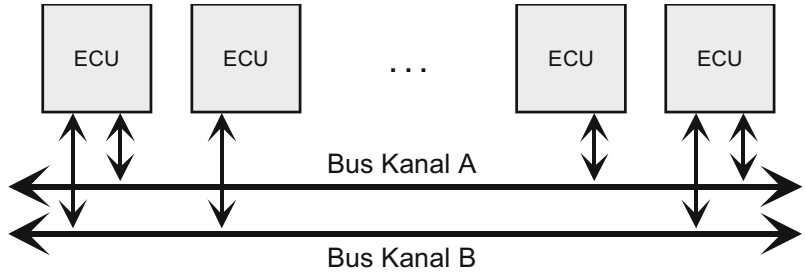


**Abb. 3.17** Beispiel eines FlexRay-Busses in Zwei-Kanal-Linien-Struktur

Scheduling in Echtzeitbetriebssystemen) trotzdem eine garantierte Maximallatenz bestimmen, doch ergeben sich dabei unter absoluten Worst Case Bedingungen oft unzulässig große Werte und die Einhaltung der Voraussetzungen ist insbesondere bei hoher Busauslastung nur schwer zu überprüfen und zu gewährleisten. Diese vor allem in der akademischen Welt diskutierte Problematik hat bei CAN zu der aufwärts kompatiblen Erweiterung Time Triggered CAN (TTCAN) geführt (siehe Abschn. 3.1.7). Das Problem der begrenzten Bit- und Datenrate dagegen konnte durch TTCAN nicht gelöst werden. Verschiedene Ansätze aus der akademischen Welt, z. B. der Time Triggered Protocol/Architecture Bus TTP/TTA, und Lösungen für spezielle Anwendungsfelder wie Byteflight, ließen eine Reihe von Konzepten entstehen, die jeweils von verschiedenen Gruppen von Fahrzeugherstellern und Zuliefern verfolgt wurden. Schließlich einigten sich die in Deutschland führenden Hersteller und Zulieferer darauf, die besten Grundideen aus diesen Ansätzen zusammenzubringen und mit FlexRay eine gemeinsame Neuentwicklung durchzuführen, die als offener, gemeinsamer Standard ähnlich wie ehedem CAN zu einer schnellen Marktdurchdringung und damit zu rasch sinkenden Kosten führen sollte. Der Spezifikationsprozess im FlexRay-Konsoritum und dann bei ISO sowie die Entwicklung entsprechender Kommunikationscontroller-ASICs nahm allerdings viel Zeit in Anspruch, so dass die Einführung von FlexRay zunächst verhalten erfolgte. Mittlerweile sind aber eine Reihe von FlexRay-Fahrzeugen im Markt, wobei FlexRay dort CAN eher ergänzt als ersetzt.

### 3.3.1 Bus-Topologie und Physical Layer

FlexRay erlaubt ein- und zweikanalige Systeme sowohl in Linien- als auch in Stern-Struktur (Abb. 3.17 und 3.18), wobei alle derzeit verfügbaren Kommunikationscontroller zweikanalig ausgeführt sind. Der maximale Abstand zwischen den am weitesten auseinander liegenden Busteilnehmern beim Linien-Bus und in der Stern-Struktur mit passivem Sternpunkt liegt bei 24 m.

Sterne mit passivem Sternpunkt haben möglicherweise schlechte elektrische Eigenschaften und sind daher nur bei kurzen Verbindungen oder niedrigen Bitraten sinnvoll.

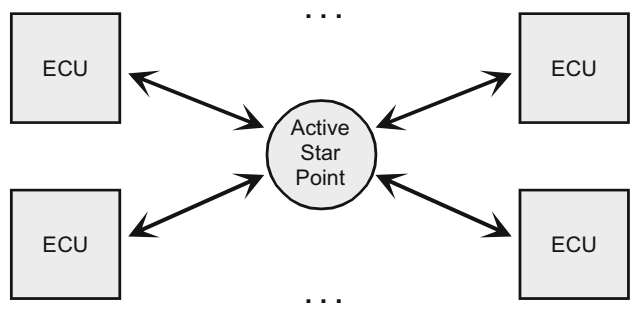


**Abb. 3.18** Beispiel eines FlexRay-Busses in Ein-Kanal-Stern-Struktur

Üblicherweise sollte der Sternpunkt deshalb mit einem aktiven Sternkoppler ausgeführt werden. Dieser wirkt als bidirektionaler Transceiver und Repeater, der die einzelnen Verbindungen elektrisch entkoppelt, aber empfangene Botschaften an alle angeschlossenen Steuergeräte weiterverteilt. D. h. logisch betrachtet arbeitet auch die Sternstruktur wie ein Bus. Mit aktivem Sternkoppler darf jedes Steuergerät bis zu 24 m vom Sternpunkt entfernt sein. Durch die Zusammenschaltung von zwei Sternpunkten, deren Direktverbindung ebenfalls max. 24 m lang sein darf, lässt sich die Ausdehnung zwischen den am weitesten entfernten Steuergeräten auf maximal 72 m steigern. Bei aktivem Sternpunkt müssen die Leitungen an beiden Enden mit Abschlusswiderständen versehen sein.

Die maximale zulässige Bitrate bei FlexRay ist gegenwärtig 10 Mbit/s. Dies wird sich bei ausgedehnten Netzwerken aber wohl nur in der Struktur mit aktivem Sternpunkt erreichen lassen. In der Linien-Struktur und bei passivem Sternpunkt wird die Leitungslänge und Teilnehmerzahl deutlich kritischer sein. Seit Protokollversion 3 sind auch Bitraten von 2,5 und 5 Mbit/s spezifiziert, die die Implementierung einfacher Linienbusse erleichtern.

Wenn das System zweikanalig, d. h. mit zwei parallelen Bussen, ausgeführt wird, kann jedes Steuergerät entweder an beide Kanäle angeschlossen werden oder auch nur an einen der beiden Kanäle (Abb. 3.17). Eine Kommunikation ist allerdings nur für die Steuergeräte möglich, die am selben Kanal angeschlossen sind, d. h. ein Steuergerät, das nur am Kanal A angeschlossen ist, kann nicht mit einem Steuergerät kommunizieren, das nur am Kanal B angeschlossen ist. Der zweite Kanal kann bei sicherheitskritischen Botschaften sowohl redundant verwendet werden, indem ein Steuergerät dieselbe Botschaft gleichzeitig auf beiden Kanälen sendet, als auch zur Bandbreitenerhöhung, indem auf beiden Kanälen unterschiedliche Botschaften versendet werden.

Theoretisch ist bei zweikanaligen Systemen auch eine gemischte (hybride) Topologie möglich, bei der z. B. Kanal A als Stern und Kanal B als Linienbus aufgebaut wird.

Die Leitungsverbindungen werden als geschirmte und/oder verdrillte Zwei-Draht-Leitungen mit einem Wellenwiderstand zwischen 80 und 110 Ω ausgeführt. An den Leitungsenden eines Linien-Bussystems bzw. an den Enden der Verbindungen zwischen Steuergerät und aktivem Sternpunkt sind wie bei High-Speed-CAN entsprechende Abschlusswiderstände notwendig. Bei passiven Stern-Strukturen sollten die Abschluss-

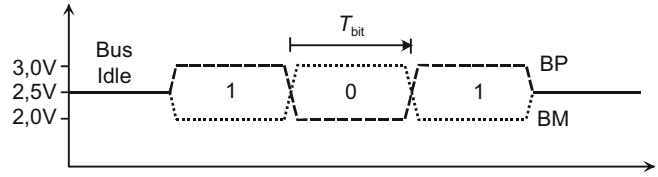


**Abb. 3.19** Signalpegel auf den FlexRay-Leitungen BP und BM

widerstände an den am weitesten voneinander entfernten Geräten angeordnet werden. Im Gegensatz zu CAN werden sowohl 0- als auch 1-Bits mit niederohmigen Differenzspannungssignalen (dominant) übertragen (Abb. 3.19). Im Ruhezustand liegen beide Busleitungen hochohmig auf ungefähr 2,5 V.

### 3.3.2 Data Link Layer

FlexRay verwendet für den Buszugriff ein ähnliches Verfahren wie TTCAN, um Kollisionen zu vermeiden. Der sich periodisch wiederholende Kommunikationszyklus wird in einen statischen und einen optionalen dynamischen Abschnitt eingeteilt, bei FlexRay statisches bzw. dynamisches Segment genannt (Abb. 3.20). Nach dem dynamischen Abschnitt folgt optional ein kurzes Zeitfenster, das so genannte *Symbol Window*, danach bleibt das Netz in Ruhe (*Network Idle Time NIT*), bevor der nächste Kommunikationszyklus beginnt. Jeder Busteilnehmer zählt die Anzahl der Kommunikationszyklen in einem Zykluszähler (*Cycle Counter*) beginnend bei 0 mit (siehe Abschn. 3.3.3). Alle Zeitabschnitte sind ganzzahlige Vielfache eines für das gesamte Netzwerk und beide Kanäle identischen virtuellen Zeitrasters, den so genannten *Makroticks*, deren Dauer zwischen 1 und 6 µs betragen soll. Das statische Segment ist für die Übertragung periodischer Botschaften vorgesehen, das dynamische Segment vorzugsweise für die ereignisgesteuerte Übertragung.

Das *statische Segment* besteht aus einer festen Anzahl von Zeitschlitzen (*Slots*), die so lang sind, dass innerhalb eines solchen Zeitschlitzes eine vollständige FlexRay-Botschaft übertragen werden kann. Die Botschaften im statischen Segment und damit die Zeitschlitze haben alle dieselbe feste Länge und laufen für beide Kanäle synchron. Das Senderecht innerhalb eines Zeitschlitzes ist für jeden Kanal getrennt genau einem einzigen Steuergerät zugeteilt (*Time Division Multiple Access TDMA*), so dass es (bei fehlerfrei arbeitenden Teilnehmern) niemals zu einer Kollision kommt. Zur redundanten Datenübertragung oder zur Bandbreitenerhöhung kann ein Steuergerät im selben Zeitschlitz auf beiden Kanälen gleichzeitig oder während mehrerer Zeitschlitze desselben Kommunikationszyklus das Senderecht erhalten. Jedes Steuergerät zählt für jeden Kanal getrennt in einem Zeitschlitz-Zähler (*Slot Counter*) beginnend bei 1 mit dem ersten Zeitschlitz jedes Kommunikationszyklus (*Cycle*) die Zeitschlitze mit. Der Stand des *Slot Counters* signalisiert im statischen

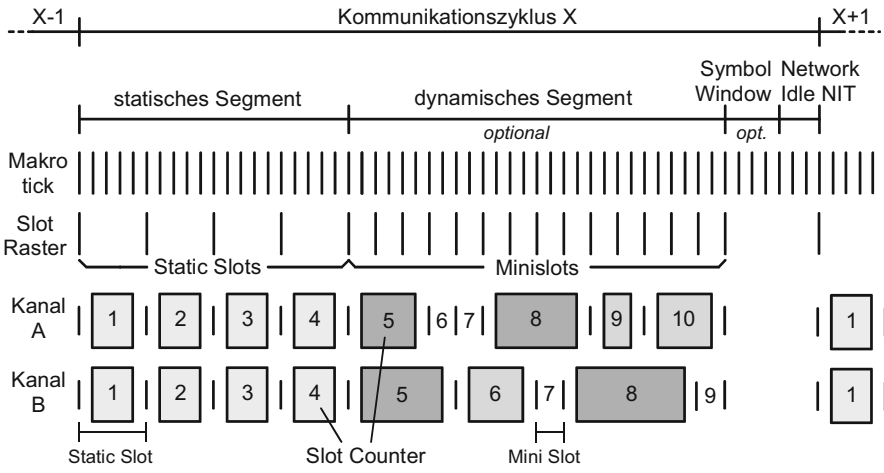


**Abb. 3.20** Ablauf der Kommunikation bei FlexRay

Segment daher, welches Gerät aktuell die Sendeberechtigung hat. Das statische Segment muss mindestens 2 und darf maximal 1023 Zeitschlitze umfassen.

Innerhalb des *dynamischen Segmentes* gibt es ebenfalls Zeitschlitze, die so genannten *Minislots*, deren Länge jedoch wesentlich kleiner ist als diejenige im statischen Segment. Während eines *Minislots* darf (für beide Kanäle unabhängig voneinander) wiederum genau ein Steuergerät senden. Die gesendete Botschaft darf jetzt aber eine beliebige, auf beiden Kanälen auch unterschiedliche Länge haben, solange ihr Ende die Gesamtlänge des dynamischen Segments nicht überschreitet. Sobald die Botschaft vollständig übertragen ist, wird der *Slot Counter* mit dem nächsten *Minislot* wieder inkrementiert und das Senderecht geht (in der Regel) auf ein anderes Steuergerät über. Hat ein Steuergerät keine Daten zu senden, verzichtet es auf das Senderecht. Der *Slot Counter* wird nach Ende einer gesendeten Botschaft oder sofort, falls keine Botschaft gesendet wurde, mit dem nächsten *Minislot* inkrementiert. Durch die variable Anzahl und Länge von Botschaften verlaufen die Zählerstände der *Slot Counter* während des dynamischen Segments in beiden Kanälen asynchron und können bei jedem Kommunikationszyklus unterschiedliche Werte annehmen (*Flexible Time Division Multiple Access FTDMA*). Innerhalb des dynamischen Segments signalisiert der *Slot Counter* daher nicht nur, welches Steuergerät die Sendeberechtigung hat, sondern indirekt auch die Priorität der zugehörigen Botschaften. Botschaften mit hohen *Slot Counter* Werten, die im dynamischen Teil des aktuellen Kommunikationszyklus nicht übertragen werden können, weil die Botschaften mit niedrigeren *Slot Counter* Werten das Segment bereits ausgeschöpft haben, müssen warten, bis sie in einem der folgenden Kommunikationszyklen übertragen werden können. Die Gesamtzahl der Zeitschlitze für den statischen und den dynamischen Teil zusammen ist auf maximal 2047 begrenzt. Alle Zeitschlitze sind ganzzahlige Vielfache der netzweiten, virtuellen *Makroticks*-Taktperiode.

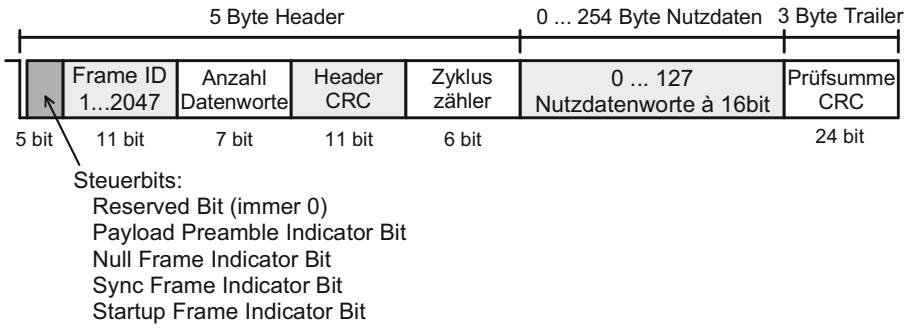


**Abb. 3.21** Logisches (*Data Link Layer*) FlexRay-Botschaftsformat

Das optionale *Symbol Window* kann zur Übertragung des sogenannten *Collision Avoidance CAS* bzw. *Media Access Test MTS* Symbols dienen, einer mindestens 30 bit langen Sequenz von Low Bits, mit denen u. a. der Buswächter (Abschn. 3.3.4) getestet werden soll. Ab Protokollversion 3 sind auch *Wakeup Pattern* im *Symbol Window* vorgesehen. In der *Network Idle Time NIT* erfolgt die Resynchronisation der Taktgeneratoren der Kommunikationscontroller (Abschn. 3.3.3).

Abbildung 3.21 zeigt den Aufbau einer einzelnen FlexRay-Botschaft. Jede Botschaft beginnt mit einem Header, der nach einigen Steuerbits im *Frame ID*-Feld die Nummer des Zeitschlitzes (*Slot*) enthält, in dem die Botschaft gesendet wird, sowie die Nutzdatenlänge. Angegeben wird dabei die Anzahl von 16 bit-Datenworten, obwohl die Daten beliebig in Bytes unterteilt sein können. Es darf aber insgesamt stets nur eine gerade Anzahl von Nutzdatenbytes, also 0, 2, 4, … usw. gesendet werden. Als letztes Feld enthält der Header die Nummer des aktuellen Kommunikationszyklus, den so genannten Zykluszähler (*Cycle*), der beim Start des Netzes mit 0 initialisiert und mit jedem Kommunikationszyklus inkrementiert wird. Sowohl der Header (ohne den Zykluszähler) als auch die gesamte Botschaft werden jeweils über eine Cyclic-Redundancy-Check-(CRC)-Prüfsumme gegen Übertragungsfehler geschützt.

Über vier der fünf Steuerbits des Headers können Sonderbotschaften angezeigt werden. Das *Payload Preamble Indicator Bit* zeigt im statischen Teil des Kommunikationszyklus an, dass die ersten 0 … 12 Datenbytes Zusatzinformationen für das Netzwerkmanagement enthalten, im dynamischen Teil, dass die ersten 2 Datenbytes eine Message ID enthalten, d. h. eine Botschaftskennung, die die weiteren Daten der Botschaft kennzeichnet und beim Empfänger ähnlich wie der Message Identifier bei CAN zur Akzeptanzfilterung verwendet werden kann. Mit dem *Null Frame Indicator Bit* kann ein Sender anzeigen, dass die Botschaft keine gültigen Nutzdaten enthält. Dies ist vor allem im statischen Teil des Kommunikationszyklus sinnvoll, wenn ein Sender zum aktuellen Zeitpunkt keine gültigen Nutzdaten hat, wegen einer gegebenenfalls im Empfänger vorhandenen Timeout-Überwachung aber trotzdem in seinem Zeitschlitz eine Botschaft senden will. Im dynamischen Teil des Kommunikationszyklus könnte der Sender in diesem Fall einfach eine Botschaft ohne Nutz-

daten senden, im statischen Teil ist dies nicht möglich, da dort alle Botschaften immer dieselbe Anzahl von Nutzdatenworten haben müssen, unabhängig davon, ob die Daten gültig sind oder nicht. Über das *Startup Frame Indicator Bit* bzw. das *Sync Frame Indicator Bit* kann eine Synchronisation aller Busteilnehmer beim Starten des Netzwerkes bzw. im laufenden Betrieb erfolgen. Details dazu im folgenden Abschn. 3.3.3.

Abbildung 3.21 zeigte den logischen Aufbau einer FlexRay-Botschaft. In der Bitübertragungsschicht werden ähnlich wie bei CAN noch einige zusätzliche Bits in den Datenstrom eingefügt, um die Bitabtastung in den Kommunikationscontrollern zu synchronisieren. Anstelle des bei CAN üblichen, aber bezüglich der Botschaftslänge nicht deterministischen Bit-Stuffings, bei dem abhängig vom Dateninhalt zusätzliche Bits übertragen werden, wird bei FlexRay vor jedem 8 bit-Datenfeld des logischen Botschaftsformats nach Abb. 3.21 zusätzlich eine 1-0-Bitfolge, die so genannte *Byte Start Sequenz BSS*, übertragen, die die effektive Datenrate um 20 % reduziert. Der Beginn der gesamten Botschaft selbst wird durch eine 3 bis 15 bit lange, als *Transmission Start Sequenz TSS* bezeichnete 0-Bitfolge und das *Frame Start Sequenz FSS Bit* eingeleitet. Die Länge der TSS wird so bemessen, dass die Transceiver eines aktiven Sternpunktes ausreichend Zeit für die Umschaltung zwischen Sende- und Empfangsrichtung haben. Dabei kann der beim Empfänger ankommende Teil der TSS dynamisch verkürzt werden. Mit einer *Frame End Sequenz FES* 0-1-Bitfolge wird die Botschaft abgeschlossen. Im dynamischen Segment folgt dann noch die *Dynamic Trailing Sequenz DTS*, die aus mindestens einem 0 und einem 1 Bit besteht und den Zeitabschnitt bis zum Beginn des folgenden *Minislots* füllt.

Da die Botschaftslänge im statischen Teil des Kommunikationszyklus für alle Zeitschlitze gleich sein muss, wird man dort wohl eher kurze Botschaften definieren. Sendet man wie bei CAN Botschaften mit max. 8 Nutzdatenbytes, so ergibt sich bei einer Bitrate von 10 Mbit/s eine maximale Nutzdatenrate für das gesamte Bussystem von max. 500 KB/s. Weil die Zeitschlitze in der Praxis aber stets etwas größer sein müssen als die Länge einer Botschaft, der Bus im dynamischen Segment wohl kaum vollständig ausgenutzt werden kann und außerdem gegebenenfalls noch Raum für das optionale *Symbol Window* und die *Network Idle Time* bleiben muss, werden die praktisch erreichbaren Werte vermutlich deutlich niedriger liegen. Im Vergleich mit einem 500 kbit/s-CAN-Bus hat ein 10 Mbit/s-FlexRay-Bussystem damit gut die zehnfache Bandbreite.

### 3.3.3 Netzwerk-Start und Takt-Synchronisation

Ein kritischer Punkt bei jedem zeitsynchronen Bussystem ist die zeitliche Synchronisation der Teilnehmer und der geordnete Start des Netzwerkes. Während bei Systemen mit asynchronem Buszugriff lediglich der Bittakt der einzelnen Teilnehmer synchronisiert werden muss, was bei CAN durch Stuffing-Bits bzw. bei FlexRay durch die *Byte Start Sequenz* BSS erreicht wird, erfordert der TDMA-Buszugriff eine Zeitsynchronisation auf der Ebene der *Makroticks* und der Zeitschlitze. Aus Zuverlässigkeitsgründen darf dabei nicht nur ein ein-

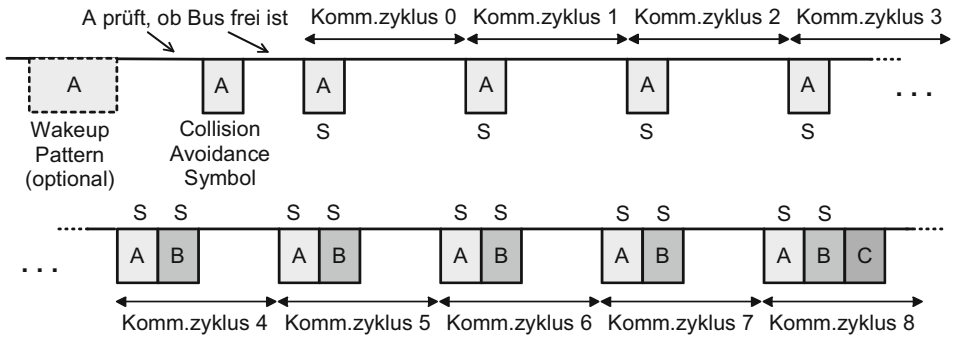


**Abb. 3.22** Hochlauf eines FlexRay Netzwerkes (TT-D Prinzip, siehe Abschn. 3.3.8), *A*: Leading Coldstart Node; *B*: Following Coldstart Node; *C*: Normales Steuergerät; *S*: Startup und Sync Frames

zelnes Steuergerät die Funktion eines Zeitmasters übernehmen, sondern es ist eine verteilte Synchronisation notwendig.

Für den Start des Netzes werden bei der Entwicklung des Netzwerks mindestens zwei, idealerweise drei Steuergeräte als Kaltstart-Knoten (*Coldstart Nodes*) festgelegt, die für den Hochlauf verantwortlich sind. Ausgelöst wird der Start entweder durch das Einschalten der Spannungsversorgung oder durch einen beliebigen Busknoten, der auf einem der beiden Bus-Kanäle ein so genanntes *Wakeup Pattern WUP* aus zwei oder mehr *Wakeup Symbolen* WUS sendet, wenn der Bus in Ruhe war.

Ein oder mehrere der Kaltstart-Knoten beginnen daraufhin, auf beiden Bus-Kanälen sogenannte *Collision Avoidance Symbole* CAS zu versenden (Abb. 3.22). Zuvor und danach überprüfen sie, ob der Bus tatsächlich frei ist, so dass im Regelfall nur ein CAS Symbol gesendet wird, weil sich die anderen Kaltstart-Knoten sofort zurückziehen, sobald sie ein fremdes CAS Symbol entdeckt haben. *Wakeup Symbol* WUS und *Collision Avoidance Symbol* CAS sind Bitmuster, die sich von allen übrigen Bitmustern auf dem FlexRay Bus eindeutig unterscheiden und damit allen angeschlossenen Steuergeräten anzeigen, dass der Kaltstart des Netzes mit dem Kommunikationszyklus 0 beginnen wird. Im Gegensatz zum *Wakeup Pattern*, das aus Sicherheitsgründen nur auf einem der beiden Bus-Kanäle versendet werden darf, muss CAS auf beiden Kanälen gleichzeitig ausgesendet werden, da die Kommunikation auf beiden Kanälen synchron initialisiert werden muss. Kaltstart-Knoten müssen also stets an beide Kanäle angeschlossen sein, während das Aufwecken der beiden Bus-Kanäle aus Sicherheitsgründen explizit von zwei verschiedenen Geräten aus erfolgen soll.

Der Hochlauf des Netzes wird fortgesetzt, indem der mit seinem CAS-Symbol erfolgreiche Kaltstart-Knoten, der so genannte *Leading Coldstart Node*, mit dem Kommunikationszyklus 0 beginnt und reguläre Botschaften innerhalb des dafür vorgesehenen Zeitschlitzes aussendet, wobei im Header dieser Botschaften stets das *Startup Frame Indicator Bit* und das *Sync Frame Indicator Bit* (Abb. 3.21) gesetzt sind. Die übrigen Kaltstart-Knoten beginnen, nachdem sie mindestens vier derartige Botschaften empfangen und sich auf das

Zeitraster synchronisiert haben (siehe unten), ebenfalls mit dem Aussenden ihrer Botschaften innerhalb der für sie reservierten Zeitschlitze, wobei auch bei deren Botschaften das *Startup Frame Indicator Bit* und das *Sync Frame Indicator Bit* gesetzt werden. Die übrigen Steuergeräte beginnen, sobald sie mindestens vier aufeinander folgende Botschaften von jeweils zwei unterschiedlichen Kaltstart-Knoten empfangen haben, ebenfalls mit dem Aussenden von Botschaften. Damit ist der Start des Netzes im günstigsten Fall nach den ersten 8 vollständigen Kommunikationszyklen abgeschlossen. Solange mindestens zwei Kaltstart-Knoten Botschaften senden, können sich andere Steuergerät jederzeit neu in die laufende Kommunikation einklinken.

Während der gesamten Kommunikation synchronisieren alle Busteilnehmer ständig ihren lokalen Takt, die so genannten *Mikroticks*, mit dem globalen Takt des Bussystems, den *Makroticks*. Dazu wertet jedes Steuergerät, auch die Kaltstart-Knoten, die Zeitabweichungen von empfangenen Botschaften mit gesetztem *Sync Frame Indicator Bit* aus und korrigiert ständig Frequenz und Phasenlage seiner eigenen Zeitbasis. Die Phasenabweichung (*Offset Error*) wird aus der Lage des Beginns der *Sync Frames* relativ zum Beginn des jeweiligen Slots bestimmt, die Frequenzabweichung (*Rate Error*) aus der zeitlichen Änderung der Phasenabweichung in aufeinanderfolgenden Kommunikationszyklen. Innerhalb des Kommunikationscontrollers werden mehrere Messwerte gespeichert. Bei jedem zweiten Kommunikationszyklus werden aus dem Mittelwert der Messwerte neue Korrekturwerte berechnet, wobei Messwertausreißer ignoriert werden (*Fault Tolerant Midpoint Algorithm*). Die Berechnung findet in jedem zweiten Zyklus im Kommunikationscontroller am Anfang der *Network Idle Time* automatisch statt. Zur Korrektur der Phasenlage wird die *Network Idle Time* dann unmittelbar um die entsprechende Zahl von *Mikroticks* verlängert oder verkürzt, während die Frequenzkorrektur durch geeignete Veränderung der *Makroticks* gleichmäßig über den gesamten Zyklus verteilt wird. Die maximale Taktfrequenzabweichung, die mit diesem Verfahren korrigiert werden kann, liegt bei 1500 ppm, d. h. etwa eine Größenordnung über den Toleranzen üblicher Quarzgeneratoren. Damit sich das System nicht aufschaukelt, wenn mehrere Knoten gleichzeitig ihre Taktperioden anpassen, muss die Empfindlichkeit des Korrekturverfahrens sorgfältig parametriert werden (*Cluster Drift Damping*). Die Messdaten und Korrekturwerte des Kommunikationscontrollers können vom steuernden Mikrocontroller gelesen und beeinflusst werden, so dass der Mikrocontroller das Bussystem softwaremäßig auch mit externen Taktquellen synchronisieren kann (*External Offset and Rate Correction*).

Innerhalb eines Netzwerkes sollen mindestens 2 und höchstens 15 Steuergeräte, die so genannten Synchronisationsknoten (*Sync Node*), im statischen Teil des Kommunikationszyklus derartige *Sync Frame* Botschaften aussenden, so dass die Synchronisation auch beim Ausfall einzelner Steuergeräte noch aufrechterhalten werden kann. Da die beiden Kanäle eines zweikanaligen FlexRay-Systems synchron arbeiten müssen, sollten diese Steuergeräte stets Botschaften mit gesetztem *Sync Frame Indicator Bit* auf beiden Kanälen senden.

Der Zeitschlitz im statischen Segment, in dem die *Sync Frame* bzw. *Startup Frame* Botschaft gesendet wird, wird auch als *Keyslot* bezeichnet. Kommunikationscontroller können so konfiguriert werden, dass sie unmittelbar nach dem Netzwerk-Start im sogenannten

*Single Slot Mode* arbeiten. In diesem Modus senden sie dann je Zyklus nur die Botschaft in ihrem *Keyslot* und verhalten sich in allen anderen Zeitschlitzen passiv. Auf diese Weise kann der Netzwerkverkehr beim Hochlauf eines Systems reduziert werden, bis alle Steuergeräte bereit sind, den Normalbetrieb aufzunehmen.

Beim fehlerhaftem Hochlauf des Netzwerks kann es zur sogenannten *Cliquenbildung* kommen. Dabei bilden sich mehrere Gruppen (*Cliquen*) von Steuergeräten, die innerhalb der jeweiligen Gruppe synchronisiert sind, zwischen den Gruppen aber asynchron bleiben. Ein derartiges Verhalten kann z. B. auftreten, wenn zwei FlexRay-Linien-Busse über einen aktiven Stern miteinander verbunden sind und der aktive Stern erst dann eingeschaltet wird, wenn sich die Steuergeräte an den beiden Linien-Bussen jeweils bereits untereinander synchronisiert haben. Cliquenbildung und andere Kommunikationsprobleme können z. B. durch Verwendung des *Network Management Vectors* erkannt werden. Wenn bei einer Botschaft im statischen Segment das *Payload Preamble Indicator Bit* gesetzt ist, werden die ersten maximal 12 Nutzdatenbytes dieser Botschaft als *Network Management Vector* interpretiert. Diese Datenbytes werden vom Kommunikationscontroller in einem speziellen Register gespeichert und dabei mit allen anderen derartigen Botschaften logisch-Oder verknüpft. Wenn jedes Steuergerät innerhalb seines *Network Management Vectors* genau ein eindeutiges Bit setzt und alle übrigen auf 0 belässt, kann jedes Steuergerät im *Network Management Register* erkennen, von welchem Steuergerät es Botschaften empfangen hat und von welchem Steuergerät nicht. Zur Auflösung von *Cliquen* muss die Kommunikation beendet und neu gestartet werden. Grundsätzlich vermeiden lässt sich *Cliquenbildung*, wenn in einem Netz nicht mehr als drei Steuergeräte für die Zeitsynchronisation konfiguriert werden.

### 3.3.4 Fehlerbehandlung, Bus Guardian

Durch die verschiedenen Zähler (*Cycle Counter*, *Slot Counter*) und Zeitüberwachungen sowie die beiden CRC-Prüfsummen erkennt der Kommunikationscontroller eine große Anzahl von Übertragungsfehlern. Abhängig von der Schwere des Fehlers schaltet er sich dann in einen passiven Modus, in dem er nicht mehr sendet, aber noch Botschaften empfängt und versucht, sich neu zu synchronisieren, oder er schaltet sich ganz ab. In jedem Fall teilt der Kommunikationscontroller der darüber liegenden Software-Protokollschicht den Fehler mit. Eine automatische Botschaftswiederholung durch die Kommunikationscontroller im Fehlerfall, wie sie bei CAN vorhanden ist, gibt es bei FlexRay nicht, um den deterministischen Betrieb auf dem Bus in jedem Fall zu gewährleisten.

In Systemen mit hohen Sicherheitsansprüchen schützt die zweikanalige Struktur gegen Ausfälle eines einzelnen Kanals, doch besteht immer noch die Gefahr, dass ein fehlerhaftes Steuergerät, das an beide Kanäle angeschlossen ist, auf beiden Kanälen unkontrolliert und zu beliebigen Zeitpunkten sendet. Ein derartiges Verhalten wird gelegentlich als *Babbling Idiot* Fehler bezeichnet. Um diesen Fehler zu beherrschen, besteht optional die Möglichkeit, zusätzlich zum Kommunikationscontroller und zu den Bustreibern in jedem Steuergerät

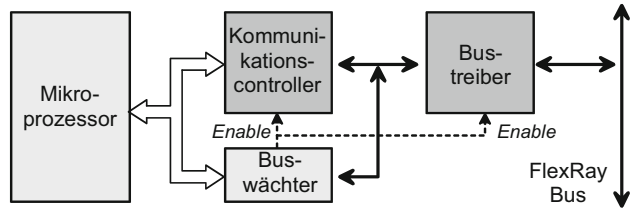


**Abb. 3.23** Kommunikationscontroller, Bustreiber und Buswächter (Darstellung bei einem einkanaligen System)

oder zentral im aktiven Sternpunkt einen so genannten Buswächter (*Bus Guardian*) einzusetzen (Abb. 3.23). Der Buswächter ist eine Art einfacher Kommunikationscontroller, der selbst keine Botschaften sendet, aber so konfiguriert ist, dass er den zeitlichen Ablauf eines Kommunikationszyklus kennt und den Sendeteil des Kommunikationscontrollers bzw. Bustreibers nur in denjenigen Zeitschlitzen freigibt, in denen das eigene Steuergerät tatsächlich senden darf. Auf diese Weise kann ein fehlerhaft arbeitendes Steuergerät zwar immer noch selbst falsche eigene Botschaften aussenden, aber nicht die Botschaften anderer Steuergeräte stören. Theoretisch sollte ein derartiger Buswächter auch schaltungstechnisch unabhängig vom Kommunikationstreiber realisiert werden, doch wird er aus Kostengründen in der Praxis wohl eher zusammen mit dem Kommunikationscontroller oder gegebenenfalls auch mit den Bustreibern integriert, so dass seine Überwachungsfunktion nur begrenzt redundant ist. Alternativ zu einem *lokalen Bus Guardian* im Steuergerät besteht die Möglichkeit, einen *zentralen Bus Guardian* z. B. im aktiven Sternpunkt anzuordnen. Die zugehörigen Spezifikationen sind in beiden Fällen jedoch ausdrücklich als vorläufig gekennzeichnet, serientaugliche Implementierungen existieren noch nicht.

### 3.3.5 Konfiguration und übergeordnete Protokolle

Ähnlich wie in der Entstehungsphase von CAN sind die unteren Protokollschichten 1 und 2 sehr detailliert, leider teilweise aber auch unübersichtlich spezifiziert. Für die höheren Protokollschichten dagegen fehlen entsprechende Festlegungen noch ganz oder werden gerade erst erarbeitet. Die Kommunikationszyklen mit der Zuordnung der Zeitschlitze zu den einzelnen Steuergeräten werden voraussichtlich überwiegend statisch, d. h. in der Entwicklungsphase konfiguriert werden. Dies gilt insbesondere auch, weil FlexRay-Kommunikationscontroller dynamisch nur umständlich und in der Regel nur mit Unterbrechung der Buskommunikation umkonfiguriert werden dürfen. Von CAN vorhandene Ansätze wie das OSEK-Netzmanagement (siehe Kap. 7) oder ein Transportprotokoll ähnlich ISO 15765-2 (siehe Kap. 4) werden in angepasster Form übernommen. Entsprechende Aktivitäten erfolgen gegenwärtig vor allem im Rahmen der AUTOSAR-Initiative (siehe Kap. 8).

Die Einführung von FlexRay verläuft langsamer als bei CAN, weil der Abstimmungsprozess der Spezifikationen in dem inzwischen viele Mitglieder umfassenden FlexRay-Konsortium naturgemäß zeitaufwendig war und sich dadurch die Bereitstellung von

**Tab. 3.10** Botschaftslängen für $f_{\text{Bit}} = 1 / T_{\text{bit}} = 10$ Mbit/s

| $n$ Data | $T$ Frame | $n$ Data | $T$ Frame |
|---|---|---|---|
| 8 byte | 17 µs | 32 Byte | 41 µs |
| 16 byte | 25 µs | 254 Byte | 263 µs |

funktionsfähigen Kommunikationscontrollern durch die Halbleiterhersteller verzögert hatte. Zum anderen erfolgt die Einführung der verteilten X-by-Wire-Anwendungen, für die FlexRay als Schlüsseltechnologie konzipiert war, aus technischen und wirtschaftlichen Gründen, die von FlexRay unabhängig sind, schleppender als erwartet.

### 3.3.6 Zeitverhalten von FlexRay-Systemen, Beispiel-Konfiguration

Die Länge einer FlexRay Botschaft (Abb. 3.21) ist näherungsweise

$$T_{\text{Frame}} \approx \left[\frac{10}{8}\left(n_{\text{Header}} + n_{\text{Trailer}} + n_{\text{Data}}\right) + n_{\text{TSS+FSS+FES}}\right] \cdot T_{\text{bit}}. \tag{3.18}$$

Dabei ist $n_{\text{Data}} = 0, \ldots, 2032$ bit die Anzahl der Datenbits, die in Stufen von 16 bit variabel ist. Header und Trailer umfassen $n_{\text{Header}} + n_{\text{Trailer}} = 64$ bit. Der Faktor 10 / 8 berücksichtigt, dass der *Physical Layer* je Byte des *Data Link Layers* zusätzlich eine *Byte Start Sequenz BSS* mit 2 bit eingefügt. Zusätzlich ergänzt der Kommunikationscontroller am Frame-Anfang eine 3 bis 15 bit lange *Transmission Start Sequenz TSS*, die Aktivierungsverzögerungen von Transceivern ausgleichen soll, sowie ein *Frame Start FES* und am Frame-Ende zwei *Frame End Sequenz FSS* Bits, insgesamt also $n_{\text{TSS+FSS+FES}} = 6 \ldots 18$ bit. In Tab. 3.10 wurde mit 10 bit gerechnet.

Beispielhaft soll ein 10 Mbit/s-FlexRay-System dargestellt werden, das sich an [11] orientiert. Die Botschaften im statischen Segment sollen $n_{\text{Data}} = 16$ Datenbytes enthalten, so dass sich eine Frame-Dauer $T_{\text{Frame}} \approx 25$ µs ergibt. Das Versenden der Botschaft wird gegenüber dem Beginn des Zeitschlitzes um den sogenannten *Action Point Offset* $T_{\text{AP,Offset}}$ verzögert und muss mindestens 11 Bitzeiten (*Channel Idle Delimiter*) vor Ende des Zeitschlitzes abgeschlossen sein (Abb. 3.24). Durch die Lücken zu Beginn und am Ende des Slots wird sichergestellt, dass der Frame aus Sicht aller Empfänger stets innerhalb desselben Slots beginnt und endet, auch wenn er durch die Bustreiber und die Leitungslaufzeit verzögert wird (*Propagation Delay,* nach Spezifikation max. 2,5 µs) und die lokalen Zeitbasen in Sender und Empfänger trotz der laufenden Taktsynchronisation voneinander abweichen (*Assumed Clock Precision* nach [14] typ. 1 … 3 µs). Im Beispiel werden die Lücken $T_{\text{AP,Offset}}$ und $T_{\text{CH,Idle}}$ zu je 5 µs gewählt, woraus sich die Gesamtdauer eines Zeitschlitzes im statischen Segment zu

$$T_{\text{StaticSlot}} = T_{\text{Frame}} + T_{\text{AP,Offset}} + T_{\text{CH,Idle}} \tag{3.19}$$

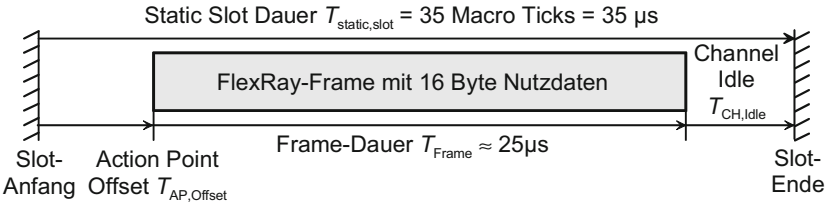


**Abb. 3.24** Aufbau eines Zeitschlitzes (*Static Slot*) im statischen Segment

ergibt, im Beispiel 35 µs. Die Länge muss ein ganzzahliges Vielfaches der Dauer eines *Makroticks MT* sein, hier wird $T_{MT}$ = 1 µs festgelegt. Laut Norm sind 1 bis 6 µs zulässig. In den *Minislots* des dynamischen Segments und im *Symbol Window* sind ähnliche Sicherheitsabstände notwendig. Da ein Frame im dynamischen Segment aber stets mehrere Minislots dauert und in unterschiedlichen *Minislots* beginnt und endet, reicht es, die Dauer der *Minislots* mit $T_{DynamicSlot} = 5\ T_{MT} = 5$ µs zu wählen.

Die interne Taktperiode $T_{\mu T}$ des Kommunikationscontrollers, mit der die Bussignale abgetastet werden, soll nach FlexRay-Spezifikation $T_{Sample} = T_{bit} / 8$ betragen, bei $f_{bit}$ = 10 Mbit/s also $T_{Sample}$ = 12,5 ns. In der Regel wird im Hinblick auf eine möglichst hohe Auflösung $T_{\mu T} = T_{Sample}$ als *Mikrotick* gewählt, obwohl auch die doppelte Dauer zulässig wäre. Im vorliegenden Beispiel ist damit $T_{MT} = 80\ T_{\mu T}$.

Innerhalb des Kommunikationszyklus mit $T_{Zyklus}$ = 5 ms Dauer wird das statische Segment mit $T_{StatSeg}$ = 3 ms festgelegt (Abb. 3.25). Insgesamt passen etwa 85 der in Abb. 3.24 dargestellten Botschaften mit je 16 Datenbyte in das statische Segment, so dass sich dafür eine Nutzdatenrate von ca. 270 KB/s ergibt. Für das dynamische Segment, das *Symbol Window SYM* und die *Network Idle Time NIT* verbleiben zusammen 2 ms. Durch die asymmetrische Aufteilung zwischen statischem und dynamischem Segment lässt sich für schnelle Regelaufgaben effektiv ein 2,5 ms Zeitraster einrichten, in dem Größen am Anfang und nochmals am Ende des statischen Segments in einem jeweils etwa 500 µs langen Fenster übertragen werden (*2,5 ms Slots*). Danach folgen Größen, die alle 5 ms übertragen werden und schließlich Signale, die nur jedes 2. Mal, jedes 4. Mal usw. gesendet werden *(Slot* bzw. *Cycle Multiplexing*) .

Mit diesem Kommunikationsschema lässt sich das statische Segment in typischen verteilten Regelsystemen einsetzen. Im einfachsten Fall werden die Regeldaten einmal je Zyklus übertragen, d. h. die Abtastzeit des Regelsystems entspricht der Zykluszeit (Fall a) in Abb. 3.26). Durch die asymmetrische Aufteilung der Segmente können die Regeldaten aber auch zweimal je Zykluszeit übertragen werden, so dass sich die Abtastzeit halbiert (Fall b). Sind Regler, Sensoren und Stellglied auf drei Steuergeräte verteilt, kann man die Totzeit im Regelkreis klein halten, indem man die Sensordaten am Anfang des statischen Segments überträgt (Fall c), so dass der Regler ausreichend Zeit hat, die Stellgröße zu berechnen und noch im selben Zyklus weiter an das Stellglied zu senden (*In Cycle Response*).

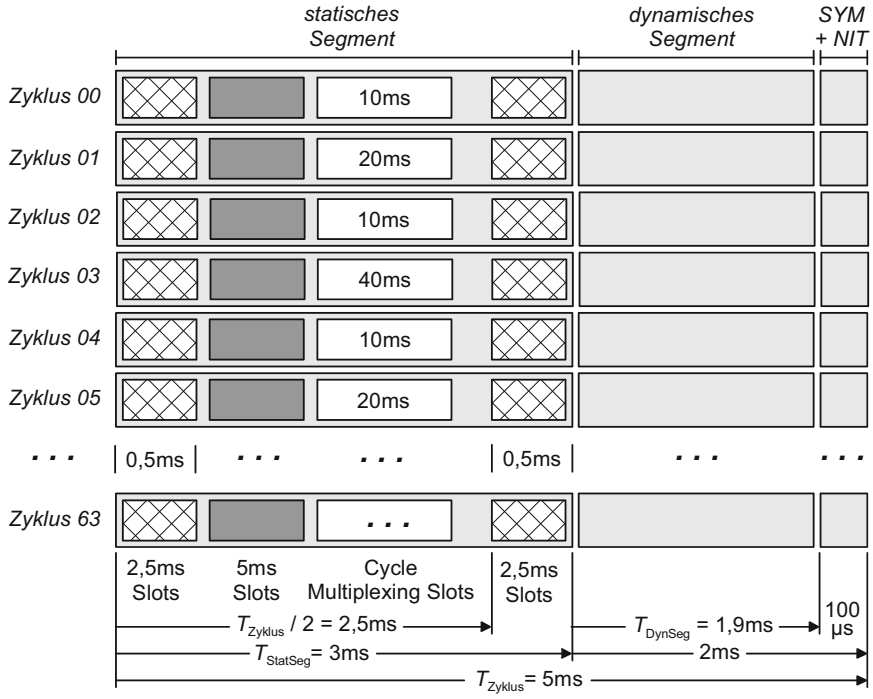


**Abb. 3.25** Kommunikationszyklen ähnlich [11]

Im dynamischen Segment können Botschaften variabler Länge versendet werden. Bei maximal 254 Byte Nutzdaten dauert ein Frame bis zu ca. 265 µs, benötigt also zusammen mit dem beschriebenen Sicherheitszuschlag ungefähr 55 Minislots. Das *Symbol Window SYM* müsste ausreichend lang für die Übertragung des *Collision Avoidance Symbols CAS* (30 bit) plus der *Transmission Start Sequenz TSS* (max. 15 bit) sein. Zusammen mit denselben Sicherheitsabständen wie im statischen Segment (Abb. 3.24) ergibt sich dafür eine Dauer von ca. 15 µs. Die *Network Idle Time NIT* muss so lang sein, dass der Kommunikationscontroller die Taktfrequenz- und Phasenkorrekturwerte auch im ungünstigsten Fall berechnen und die Phasenkorrektur ausführen kann, wobei die Berechnung bereits während des dynamischen Segments beginnen darf. Im Beispiel werden für das *Symbol Window* und die *Network Idle Time* zusammen ca. $T_{SW} + T_{NIT} = 100$ µs reserviert. In das dynamische Segment mit einer Länge von $T_{DynSeg} = T_{Zyklus} - T_{StatSeg} - (T_{SW}+T_{NIT}) = 1{,}9$ ms passen damit gut 7 Botschaften mit 254 Byte Nutzdaten, woraus sich im günstigsten Fall eine Nutzdatenrate von etwa 350 KB/s ergibt. Sendet man auch hier lediglich Botschaften mit im Mittel nur 16 Byte Nutzdaten, kann man zwar über 50 solcher Botschaften übertragen, die effektive Datenrate für das dynamische Segment reduziert sich dann aber auf unter 180 KB/s.

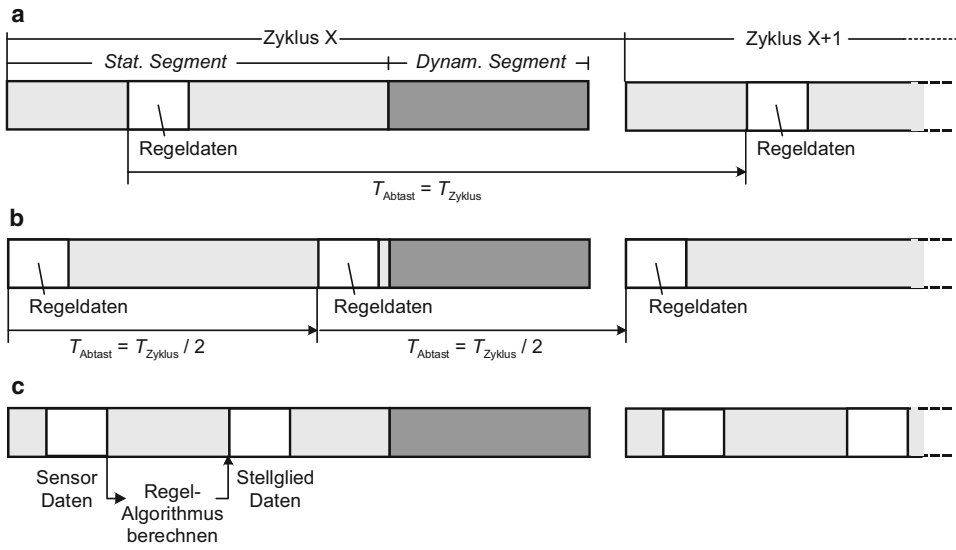


**Abb. 3.26** Nutzung des statischen Segments in verteilten Regelsystemen

**Tab. 3.11** FlexRay-Parameter bei BMW nach [11]

| | | | |
|---|---|---|---|
| Zykluszeit | 5 ms | Dauer Makrotick | 1,375 µs |
| Statisches Segment | 3 ms | Dynam. Segment inkl. NIT | 2 ms |
| Nutzdaten im stat. Segment | 16 Byte | Nutzdaten im dyn. Segment | 2 … 254 Byte |
| Statische Slots | 91 | Minislots (Dynamische Slots) | 289 |
| Dauer stat. Slot | 33 µs | Dauer Minislot | 6,875 µs |

Zum Vergleich mit dem obigen Beispiel zeigt Tab. 3.11 die FlexRay-Parameter, wie sie bei BMW im Serieneinsatz sind.

#### 3.3.6.1 Latenzzeiten im statischen und im dynamischen Segment

Für die Latenzzeit von Botschaften im statischen Segment gelten dieselben Überlegungen wie bei LIN (Abschn. 3.2.8). Es ergibt sich also

$$T_{\text{Frame}} < T_{\text{Latenz,StatSeg}} < T_{\text{Zyklus}} + T_{\text{Frame}}. \tag{3.20}$$

Um den Jitter möglichst gering zu halten, sollte die Anwendungssoftware, welche die Daten bereitstellt, mit dem Zyklus des Bussystems synchronisiert werden.

Im dynamischen Segment kann eine Botschaft in einem späteren Minislot durch Botschaften in früheren Minislots verzögert werden. Ist die Verzögerung so groß, dass die Botschaft nicht mehr sicher bis zum Ende des aktuellen dynamischen Segments übertragen werden kann, so wird die Botschaft für einen oder gar mehrere Zyklen verzögert. Die Zuordnung einer Botschaft zu einem bestimmten Minislot wirkt damit wie die Festlegung

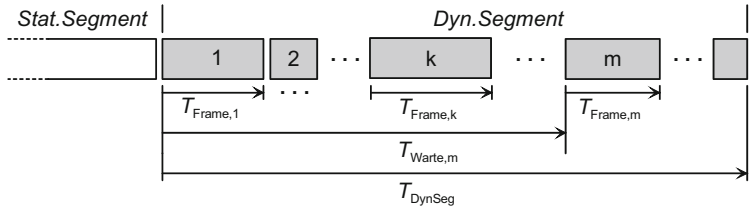


**Abb. 3.27** Botschaften im dynamischen Segment

der Priorität einer Botschaft. Für die Berechnung der Latenz werden folgende Annahmen getroffen:

- Die Botschaft wird am Anfang des dynamischen Segments sendebereit. Die Latenzzeit bezieht sich auf diesen Zeitpunkt.
- Die Minislots werden beginnend mit $k = 1$ durchnummeriert, die betrachtete Botschaft soll im Minislot $m$ gesendet werden. Die Dauer eines Minislots sei $T_{\text{DynamicSlot}}$.
- Die Botschaften des dynamischen Segments werden periodisch mit der jeweiligen Periode $T_k$ versendet. In der Regel wird $T_k > T_{\text{Zyklus}}$ sein, sonst könnte man die Botschaften ja auch im statischen Segment versenden. Für Botschaften, die nicht periodisch versendet werden, wird $T_k$ als Mindestabstand zwischen zwei Übertragungen interpretiert (*Interarrival Time*).

Die Latenzzeit der Botschaft $m$ setzt sich aus der Wartezeit bis zum Beginn der Übertragung und der eigentlichen Übertragungsdauer zusammen (Abb. 3.27):

$$T_{\text{Latenz,DynSeg},m} = T_{\text{Warte},m} + T_{\text{Frame},m}. \tag{3.21}$$

Die kleinstmögliche Wartezeit für die Botschaft $m$ ergibt sich, wenn vor ihr im dynamischen Segment keine anderen Botschaften gesendet werden. Dann muss die Botschaft nur die $(m-1)$ leeren Minislots abwarten, bevor die Übertragung beginnt:

$$T_{\text{Warte},m,\text{min}} = (m-1) \cdot T_{\text{DynamicSlot}}. \tag{3.22}$$

Am größten wird die Latenzzeit für die Botschaft m, wenn in den vorherigen $(m-1)$ Minislots des dynamischen Segments auch andere Botschaften gesendet werden. Für diesen Fall kann die Wartezeit abgeschätzt werden:

$$T_{\text{Warte},m} \approx \left( \sum_{k=1}^{m-1} T_{\text{frame},k} \right) \text{ modulo } T_{\text{DynSeg}} + \left\lfloor \frac{\sum_{k=1}^{m-1} \left\lceil \frac{T_{\text{Warte},m}}{T_k} \right\rceil T_{\text{Frame},k}}{T_{\text{DynSeg}}} \right\rfloor \cdot T_{\text{Zyklus}}. \tag{3.23}$$

Der erste Term beschreibt die Verzögerung durch diejenigen Botschaften, die im selben dynamischen Segment vor Botschaft m gesendet werden. Für nicht belegte Minislots wird $T_{\text{Frame,k}} = T_{\text{DynamicSlot}}$. Der zweite Term ist zu beachten, wenn die Botschaften $k = 1 \ldots m$ nicht mehr alle in ein einziges dynamisches Segment passen, d. h. wenn

$$\sum_{k=1}^{m} T_{\text{Frame},k} > T_{\text{DynSeg}} \tag{3.24}$$

ist. In diesem Fall vergrößert sich die Wartezeit um einen oder sogar mehrere Zyklen. Die Anzahl der zusätzlichen Zyklen wird durch den Term in $\lfloor \ldots \rfloor$ beschrieben, der auf den nächsten ganzzahligen Wert abgerundet wird. Der Ausdruck in $\lceil \ldots \rceil$, der auf den nächsten ganzzahligen Wert aufgerundet werden muss, berücksichtigt, dass die anderen Botschaften während einer längeren Wartezeit mehrfach sendebereit werden können. Ähnlich wie bei CAN (Abschn. 3.1.7) ist auch diese Gleichung nur iterativ lösbar. Gleichung 3.23 liefert nur eine grobe Abschätzung, weil sie nicht berücksichtigt, dass nicht nur Botschaft m sondern auch alle anderen Botschaften stets vollständig innerhalb des laufenden Zyklus gesendet werden müssen. Andernfalls müssen sie ebenfalls für mindestens einen Zyklus warten. Dadurch kann sich sogar die Reihenfolge ändern, wenn die Botschaft in einem früheren Minislot nicht gesendet werden kann, weil sie nicht mehr in das aktuelle Segment passt, während eine eigentlich erst in einem späteren Minislot folgende, aber kürzere Botschaft noch gesendet werden kann. In den einzelnen Zyklen bleiben dann unterschiedliche Minislots frei. Die exakte Berechnung des Worst Case Falles ist aufwendig und wird z. B. in [16] dargestellt.

Gleichung 3.24 unterteilt das dynamische Segment faktisch in eine Klasse von Botschaften, deren Übertragungsverzögerung trotz der ereignisgesteuerten Übertragung im dynamischen Segment deterministisch bleibt, und eine zweite Klasse, deren Übertragungsverzögerung extrem stark schwanken und für die nur schwer eine Obergrenze angegeben werden kann.

### 3.3.7 Schnittstelle zum FlexRay-Controller

Bevor ein FlexRay-Kommunikationscontroller (Abb. 3.28) Botschaften senden und empfangen kann, muss er zunächst konfiguriert werden. Die FlexRay-Spezifikation definiert dafür einen Zustandsautomaten *Protocol Operation Control POC* (Abb. 3.29) und legt die Befehle fest, mit denen der steuernde Mikrocontroller die Zustandsübergänge über das *Controller Host Interface CHI* auslöst.

Nach dem Einschalten (*Reset*) werden über die Protokollkonfigurationsregister zunächst die Bitrate, die Dauer von Makro- und Mikroticks sowie die Längen von statischem und dynamischem Segment, *Symbol Window* und *NIT* festgelegt. Diese in der FlexRay-Spezifikation beschriebenen mehr als 50 Konfigurationsparameter dürfen nur im Zustand *Config* verändert werden, in dem noch keine Buskommunikation stattfindet. Anschlie-

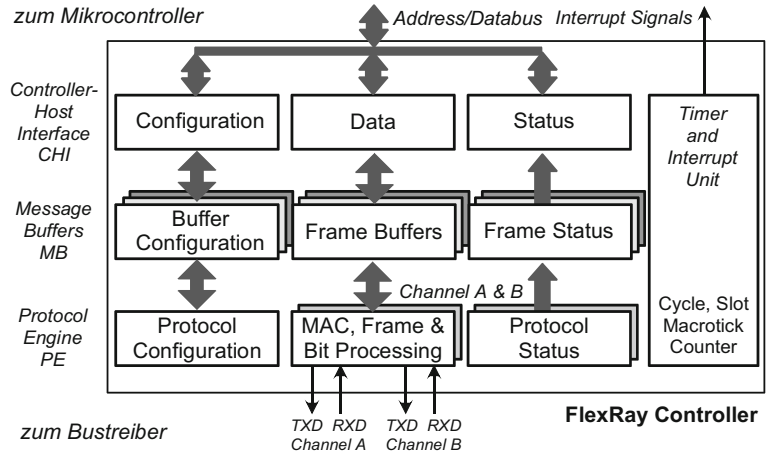


**Abb. 3.28** Blockschaltbild eines typischen FlexRay-Kommunikationscontrollers

ßend konfiguriert der Mikrocontroller die Botschaftsspeicher (*Message Buffer*), bevor er die Kommunikation über den Zwischenzustand *Ready* und den Befehl *Run* freigibt. Der Kommunikationscontroller führt dann selbstständig die in Abschn. 3.3.3 beschriebene Integration in eine bestehende Buskommunikation durch bzw. beteiligt sich aktiv am Start des Netzwerkes, falls er zuvor mit dem Befehl *Allow Coldstart* als Kaltstart-Knoten

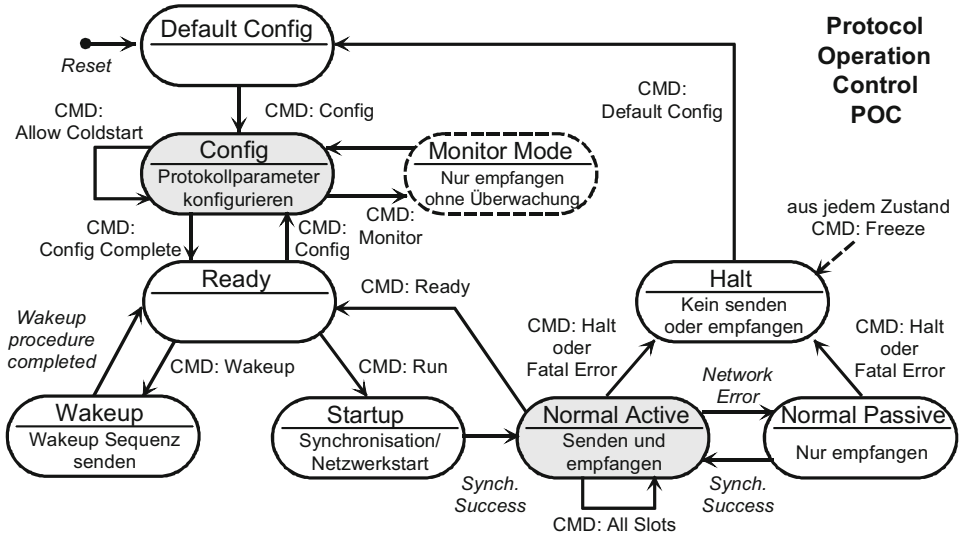


**Abb. 3.29** *POC* Zustände des FlexRay-Kommunikationscontrollers (vereinfacht), *CMD*: Host Command des steuernden Mikrocontrollers; *kursiv*: automatisch erfolgende Zustandsübergänge

konfiguriert wurde. Nach erfolgter Synchronisation wechselt er in den Zustand *Normal Active*, in dem das vorkonfigurierte Senden und Empfangen von Botschaften erfolgt. Falls der Kommunikationscontroller so konfiguriert war, dass er zunächst im *Single Slot Mode* arbeitet, wird er durch den Befehl *Allow All Slots* in den Normalbetrieb umgeschaltet, in dem er in allen konfigurierten Zeitschlitzen senden darf. Beim Auftreten von Synchronisationsfehlern wechselt er je nach Schwere des Fehlers in den Zustand *Normal Passive*, in der er nur noch Botschaften empfängt, oder in den Zustand *Halt*, den er nur durch eine Neukonfiguration wieder verlassen kann. Bei Bedarf kann der Mikrocontroller die Kommunikation anhalten und den Kommunikationscontroller in den Zustand *Ready* versetzen, in dem keine Botschaften versendet oder empfangen werden, sondern lediglich der Empfang eines *Wakeup Patterns WUP* erkannt wird. Von dort aus kann der Mikrocontroller entweder selbst das Versenden eines *Wakeup Patterns* veranlassen und/oder die Kommunikation über den Befehl *Run* wieder aufnehmen. Zusätzlich lassen sich viele FlexRay-Kommunikationscontroller in einen *Monitor Mode* versetzen, in dem Botschaften und Symbole empfangen werden, ohne dass der Kommunikationscontroller sich auf das Slot- und Zyklusraster aufsynchronisieren muss. Dieser Modus kann ähnlich wie der *Listen* oder *Silent Mode* bei CAN-Kommunikationscontrollern (siehe Abschn. 3.3.6) zu Diagnosezwecken etwa während des Netzwerk-Starts eingesetzt werden.

Ebenso wie die CHI-Befehlsschnittstelle gibt die FlexRay-Spezifikation auch vor, welche Statusinformationen der Controller bereitstellen muss. Zu den über 40 geforderten Werten gehören neben dem aktuellen Makrotick, dem Zykluszähler und den *Slot Countern* für die beiden FlexRay-Kanäle Informationen über die Zeitsynchronisation und empfangene Symbole, Fehlerflags der verschiedenen Format- und Zeitüberwachungsmechanismen sowie der *Network Management Vector*.

Im Gegensatz zur Konfiguration, Ablaufsteuerung und Statusabfrage des Protokolls definiert die FlexRay-Spezifikation den Aufbau der Botschaftsspeicher (*Message Buffers*) nur relativ grob (Abb. 3.30). Daher sind die Implementierungsunterschiede zwischen verschiedenen Herstellern verhältnismäßig groß. In jedem Fall kann man den oder die FlexRay-Kanäle und die *Frame* bzw. *Slot ID* konfigurieren und einstellen, ob die Botschaft bei jedem Kommunikationszyklus oder nur bei jedem 2., 4. Mal usw. gesendet bzw. empfangen werden soll. Zum Teil wird zur Unterstützung des *Slot Multiplexing* eine Empfangsfilterung auf Basis der 16 bit großen *Message ID* angeboten, die am Anfang einer Botschaft im dynamischen Segment versendet werden kann. Ansonsten findet die Sendeentscheidung bzw. Empfangsfilterung auf Basis der Kanal-, Zyklus- und Slot-Zähler-Konfiguration statt. Wie bei CAN-Controllern lässt sich die Empfangsfilterung meist auch so konfigurieren, dass ein Botschaftsspeicher für eine Gruppe vor Botschaften gemeinsam verwendet werden kann.

Zum Senden muss die Anwendung auf dem Mikrocontroller den kompletten *Frame Header* sowie die Nutzdaten bereitstellen. Im statischen Segment reicht in der Regel eine einmalige Konfiguration des *Frame Headers*. Im dynamischen Segment dagegen muss die im *Frame Header* enthaltene CRC-Prüfsumme vom Mikrocontroller jedes Mal neu berechnet werden, wenn sich die Nutzdatenlänge ändert (Abb. 3.21). Der Zykluszähler, der nicht Teil der *Header*-Prüfsumme ist, sowie die CRC-Prüfsumme im *Frame Trail*er werden vom

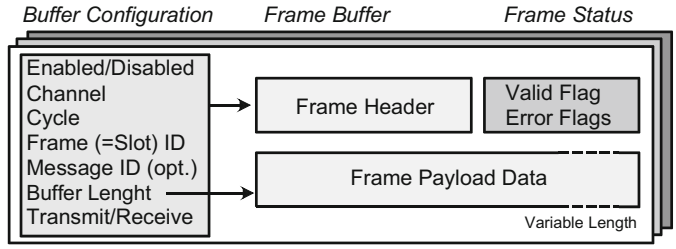


**Abb. 3.30** Typischer FlexRay-Botschaftsspeicher (*Message Buffer*)

Kommunikationscontroller automatisch ermittelt. Sobald die Daten komplett in den Puffer kopiert wurden, zeigt der Mikrocontroller dies durch Setzen des *Valid Flags* an. Nach dem tatsächlichen Versenden der Botschaft informiert der Kommunikationscontroller den steuernden Mikroprozessor durch Setzen eines Flags oder durch einen Interrupt. Optional setzt der Kommunikationscontroller das *Valid Flag* nach dem Senden automatisch zurück, so dass stets nur Daten versendet werden, die vom Mikrocontroller aktualisiert wurden (*Event Triggered Transmit*). Befinden sich zum Sendezeitpunkt keine gültigen Daten im *Frame Buffer*, sendet der Kommunikationscontroller im statischen Segment einen *Null Frame*, im dynamischen Teil sendet er dann gar keine Botschaft. Analog setzt der Kommunikationscontroller das *Valid Flag* beim Empfang einer Botschaft und informiert den Mikrocontroller optional per Interrupt über den Empfang. Sowohl beim Versenden als auch beim Empfang prüft der Kommunikationscontroller, ob *Header*-Format sowie Prüfsummen korrekt sind und die Zeitbedingungen bezüglich *Frame*-Beginn und Ende eingehalten wurden und zeigt Fehler durch verschiedene *Error Flags* an.

Weiterhin fordert die Spezifikation die Bereitstellung von mindestens zwei absolut bzw. relativ arbeitenden *Timern*, die den Mikrocontroller beim Erreichen bestimmter Stände des Zyklus-, Makrotick- und/oder *Slot Counters* durch Interrupts alarmieren können, so dass die dort ablaufende Software sich mit dem Bussystem synchronisieren und die Sendebotschaften rechtzeitig bereitstellen kann.

Die Anzahl und Größe der Botschaftsspeicher sowie der Mechanismus, mit dem der Zugriff des Mikrocontrollers und der *Protocol Engine* auf den Botschaftsspeicher synchronisiert werden, sind implementierungsabhängig. Der in [14] beschriebene Kommunikationscontroller beispielsweise kann zwischen 128 Botschaften mit je 48 Byte Nutzdaten und 30 Botschaften mit je 254 Byte Nutzdaten speichern. Ein Teil des Botschaftsspeichers lässt sich alternativ als FIFO konfigurieren, so dass auch Botschaften empfangen werden können, für die kein eigener Botschaftsspeicher bereitgestellt werden kann. Für die Synchronisation beim Zugriff auf den Botschaftsspeicher können alternativ sowohl aufwendigere Doppel-Puffer-Konzepte als auch einfachere Lock/Unlock-Mechanismen implementiert werden [15]. Bei Doppel-Puffer-Konzepten kann der Mikrocontroller einen Botschaftspuffer auslesen, während die *Protocol Engine* bereits eine neue Botschaft in den zugeordneten Hintergrundspeicher schreibt und umgekehrt. Beim Lock/Unlock-Verfahren sperrt

die jeweils zugreifende Einheit den Zugriff für die andere Einheit, so dass zumindest keine inkonsistenten Botschaften gelesen oder gesendet werden, gegebenenfalls aber Botschaften verloren gehen.

Sollen mehr Botschaften gesendet oder empfangen werden als *Message Buffer* zur Verfügung stehen, muss die Treibersoftware die *Buffer* dynamisch umkonfigurieren. Dies ist eine verhältnismäßig anspruchsvolle Operation. Falls ein Empfangs- oder Sendepuffer zu spät aktiviert wird, geht die Botschaft unter Umständen verloren. Wird ein Sendepuffer zu früh aktiviert oder zu spät deaktiviert, kann es sogar zu Kollisionen auf dem Bus kommen, wenn derselbe Zeitschlitz im dynamischen Segment im Multiplex-Betrieb auch von einem anderen Busteilnehmer genutzt wird. Gerade die zeitliche Synchronisation der Treibersoftware im dynamischen Segment ist außerordentlich kritisch, weil die exakte zeitliche Lage eines *Slots* dort davon abhängt, welche Botschaften vorher gesendet oder nicht gesendet wurden. Unter Umständen kann sogar zu einer Verschiebung in einen der folgenden Kommunikationszyklen kommen, falls das Segment bereits durch vorherige Botschaften zu lange belegt war.

### 3.3.8 Weiterentwicklung FlexRay 3.x

Nachdem das Protokoll mit Version 2.1 A einen stabilen Zustand erreicht hatte, eine Reihe von Kommunikationscontrollern auf dem Markt und erste Fahrzeuge mit FlexRay im Serieneinsatz waren, sollte mit der Ende 2010 veröffentlichten Version 3.0 offene Punkte in der Protokollspezifikation geklärt werden. Die Randbedingungen und gegenseitigen Abhängigkeiten der Parameter eines FlexRay-Systems wurden ausführlicher spezifiziert, doch gibt es leider weiterhin keinen Default-Parametersatz. Die Inbetriebnahme erfordert daher immer noch umfangreiche Konfigurationsarbeiten, die für Entwickler ohne umfangreiche FlexRay-Erfahrung und Werkzeugunterstützung fehlerträchtig und mühsam sein können.

Kleinere Änderungen betreffen das sogenannte *Cycle* bzw. *Slot Multiplexing*, d. h. die Zuteilung der Sendeberechtigung in einem Zeitschlitz auf verschiedene Steuergeräte in unterschiedlichen Kommunikationszyklen. Als Periodendauern für das *Multiplexing* sind nicht mehr nur duale (1, 2, 4, …) sondern auch dezimale Vielfache (5, 10, …) von $T_{\text{Zyklus}}$ zulässig. Der Zykluszähler, der bisher stets 64 Kommunikationszyklen durchlief, darf bei jedem geraden Wert zwischen 8 und 64 vorzeitig zurückgesetzt werden. Außerdem ist das *Slot Multiplexing* nun nicht nur im dynamischen, sondern auch im statischen Segment erlaubt.

Des weiteren spezifiziert FlexRay 3.0 zusätzlich zu 10 Mbit/s auch die Bitraten 2,5 und 5 Mbit/s vollständig. Dadurch lassen sich EMV-Probleme reduzieren und FlexRay-Systeme in kostengünstigerer Linienbus-Topologie aufbauen, weil eventuell auf den teuren aktiven Sternpunkt verzichtet werden kann. Diese Änderungen wurde u. a. vom JASPAR-Konsortium angeregt, das die Übernahme der FlexRay- und AUTOSAR-Spezifikationen für die Anwendung bei japanischen Herstellern prüft.

**Tab. 3.12** Konfiguration bei verschiedenen Netzwerk-Start Konzepten

| | TT-D | TT-L | TT-E |
|---|---|---|---|
| Anzahl *N* der *Coldstart Sync Nodes* | 2 … 15 | 1 | 1 … 7 |
| Anzahl *M* der *Non-Coldstart Sync Nodes* | 0 … 15 – *N* | 0 | 0 |
| Anzahl der *Keyslots* je *Coldstart Node bzw. Non-Coldstart Sync Node* | 1 | 2 | 2 |
| Anzahl der *Keyslots* bei den anderen Steuergeräten | 0 | 0 | 0 |

Weitere Neuerungen gibt es im Bereich des Netzwerk-Starts und der Takt-Synchronisation. Beim bisherigen, zur Unterscheidung als *Time Trigger – Distributed TT-D* bezeichneten Verfahren sind mindestens zwei Steuergeräte als *Coldstart Nodes* notwendig (Abschn. 3.3.3). Diese Geräte senden in jeweils genau einem Zeitschlitz je Steuergerät, dem *Keyslot*, Botschaften mit gesetzten *Startup Frame* und *Sync Frame Indicator Bit*. Mit Hilfe dieser Botschaften wird das Netz im günstigsten Fall innerhalb von 8 Kommunikationszyklen gestartet und erhält im laufenden Betrieb die zeitliche Synchronität aufrecht (Abb. 3.22). Um die Verfügbarkeit des Netzes bei Ausfall einzelner Geräte zu erhöhen, können insgesamt bis zu 15 Steuergeräte als *Coldstart Nodes* konfiguriert werden (Tab. 3.12). Zusätzlich zu den *Coldstart Nodes* dürfen weitere Steuergeräte als *Non-Coldstart Sync Nodes* betrieben werden. Diese Steuergeräte senden in ihrem jeweiligen Keyslot Botschaften mit aktivem *Sync Frame*, aber inaktivem *Startup Frame Indicator Bit*. Dadurch tragen sie zur Taktsynchronisation bei, können das Netz aber nicht selbst starten.

Mit Protokollversion V3.0 neu eingeführt wurde das Verfahren *Time Trigger – Local Master TT-L*. Mit *TT-L* sind einfachere Netze möglich, bei denen nur noch ein einziges Steuergerät, der *Local Master*, für den Netzwerk-Start und die Zeitsynchronisation verantwortlich ist. Dieses Gerät sendet in zwei *Keyslots* Botschaften mit gesetzten *Startup Frame* und *Sync Frame Indicator Bits*. Für alle anderen Steuergeräte erscheint der Netzwerk-Start wie bei TT-D (Abb. 3.22). Da bei *TT-L* der Hochlauf des bei *TT-D* notwendigen zweiten Coldstart-Knotens entfällt, braucht das *TT-L* Netz im günstigsten Fall nur 6 statt 8 Kommunikationszyklen, bis das Netz gestartet ist.

Das neue *Time Trigger – External TT-E* Konzept soll komplexe Systeme ermöglichen, bei denen mehrere FlexRay-Busse synchron zueinander betrieben werden müssen. Eines der Bussysteme, die *Time Source*, gibt den Kommunikationszyklus vor und wird als klassisches *TT-D* Netz betrieben (Abb. 3.21). Das *TT-E* Bussystem, die *Time Sink*, wird über mindestens ein als *Time Gateway* wirkendes Steuergerät mit zwei FlexRay-Schnittstellen angekoppelt. Bezüglich des *Time Source* Netzes stellt das *Time Gateway* einen normalen FlexRay-Knoten mit oder ohne *Coldstart* und *Sync* Eigenschaften dar. Im *Time Sink* Netz dagegen übernimmt das *Time Gateway* die Rolle des *Coldstart* und *Sync* Knotens. Für die anderen Steuergeräte im *Time Sink* Netz wirkt der *TT-E* Netzwerk-Start wie bei einem *TT-L* Netz. Das *TT-E* Netz läuft bezüglich Kommunikationszyklen und *Makrotick* Takt mit einem festen Offset von 40 *Mikroticks* synchron zum *Time Source* Netz. Um die Verfügbarkeit

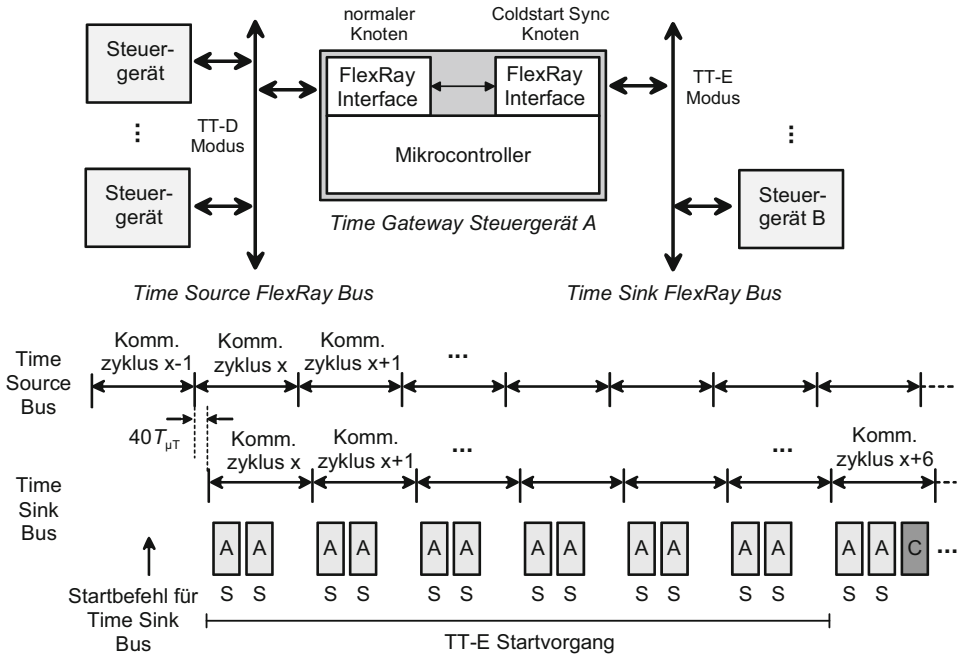


**Abb. 3.31** Startvorgang eines TT-E FlexRay Netzes (S … Startup und Sync Frame)

zu erhöhen, können auch mehrere Steuergeräte als *Time Gateways* zum selben *Time Source* Netz eingesetzt werden.

### 3.3.9 Zusammenfassung FlexRay – Layer 1 und 2

- Kommunikation zwischen mehreren Kfz-Steuergeräten zum Austausch von Mess-, Steuer- und Reglersignalen im Echtzeitbetrieb mit hoher Fehlersicherheit.
- Aufgrund der hohen Bandbreite auch als Backbone-Netz zu Kopplung von Gateways zwischen verschiedenen Teilnetzen einsatzbar.
- Maximal 64 Steuergeräte je Bussegment.
- Bitstrom-orientiertes Übertragungsprotokoll mit bidirektionaler Zwei-Draht-Leitung als Linien-Bus oder in Sternstruktur. Buslänge bis zum Sternpunkt max. 24 m.
- Ein- und zweikanalige Systeme möglich. Zweiter Kanal für redundante Übertragung in sicherheitskritischen Anwendungen oder zur Erhöhung der Bandbreite.
- Bitraten 2, 5 und 10 Mbit/s, theoretisch zukünftig auch höhere Bitraten möglich.
- FlexRay-Controller, Transceiver und Mikroprozessor notwendig. Buswächter (Bus Guardian) optional.

- Broadcast-System, bei dem die Sendeberechtigung und die Nachrichten festen Zeitschlitzen innerhalb eines Kommunikationszyklus zugewiesen sind. Im statischen Teil des Kommunikationszyklus zeitsynchroner (TDMA) Buszugriff mit Zeitschlitzen fester Länge, im dynamischen Teil modifiziertes TDMA-Verfahren (FTDMA) mit variablen Zeitschlitzen und positionsabhängiger Arbitrierung (Teilnehmer mit früherem Zeitschlitz kann Teilnehmer mit späterem Zeitschlitz blockieren).
- Botschaften mit 0…254 Datenbytes und 8 Byte Header/Trailer. Zusätzliche Steuerbits auf der Bitübertragungsebene mit ca. 20 % Overhead. Nutzbare Datenrate < 500 … 1000 KB/s je Kanal (bei 10 Mbit/s). Datenübertragung durch CRC-Prüfsummen gesichert, aber keine automatische Sendewiederholung bei Fehlern.

## 3.4 Media Oriented Systems Transport MOST

Während die bisher diskutierten Bussysteme alle für Steuer- und Regelaufgaben entwickelt wurden (*Control Bus*), ist MOST für Telematik- und Multimedia-Anwendungen (*Infotainment Bus*) konzipiert, d. h. für die Vernetzung von Autoradio, DVD-Wechsler, Autotelefon, Navigationssystem und Bordfernsehgerät [17, 18]. Bei diesen Anwendungen sind die Anforderungen an Echtzeitverhalten und Übertragungssicherheit geringer, die notwendigen Übertragungsbandbreiten aber höher als bei typischen Steuer- und Regelaufgaben. Um Sprache in Telefonqualität übertragen zu können, ist bei unkomprimierter Übertragung eine Nutzdatenrate von min. 8 KB/s notwendig, für nicht komprimierte Musiksignale in Stereo-CD-Qualität (zwei Kanäle mit 16 bit Abtastwerten bei 44,1 kHz Abtastfrequenz) min. 176 KB/s, also deutlich mehr als etwa mit CAN möglich ist. Komprimierte Audiosignale kommen bei Stereo mit 16 KB/s (MP3) bis 24 KB/s (AC3) aus, während für *Dolby Surround Sound* (AC3 5.1) bis zu 56 KB/s notwendig werden. Nach MPEG-1 bzw. -2 komprimierte Videosignale, wie sie z. B. auf DVDs üblich sind, benötigen bei Standard-Fernsehbildqualität um 1,5 MB/s.

Ziel eines Infotainment-Bussystemes ist es, Audio- und Videodaten digital und damit störunempfindlich zwischen verteilten Geräten zu übertragen. Als direkter Vorgänger von MOST gilt dabei der von der Firma Philips entwickelte Domestic Data Bus D2B. D2B kam jedoch nur in wenigen Fahrzeugen zum Einsatz, seine Weiterentwicklung wurde eingestellt. Neben D2B gab es anfänglich auch Systeme, die sich mit dem SPDIF Standard aus der Unterhaltungselektronik beholfen haben. Aufgrund der fehlenden Möglichkeit, mehrere Komponenten mit SPDIF intelligent zu verbinden, konnte sich dieser Ansatz aber nicht durchsetzen.

MOST war wie D2B ursprünglich die Entwicklung eines einzelnen Herstellers, in diesem Fall der Firma Oasis Silicon Systems (später SMSC, heute Microchip). Als die Weiterentwicklung von MOST auf Betreiben einiger Automobilhersteller 1998 einem Firmenübergreifenden Konsortium, der *MOST Cooperation*, übertragen wurde, setzte es sich rasch in europäischen Oberklassefahrzeugen durch, andere Hersteller reagierten zunächst zurückhaltend, setzen mittlerweile aber in einzelnen Fahrzeugen ebenfalls MOST ein. Dabei

spielt neben den guten Eigenschaften des eigentlichen Bussystems sicher eine wesentliche Rolle, dass MOST eine Spezifikation für die gesamten ISO/OSI-Schichten 1 bis 7 (siehe Kap. 2) bietet und frühzeitig funktionsfähige Buscontroller und Transceiver verfügbar waren. Die MOST-Kommunikationscontroller implementieren zusammen mit den zugehörigen Bus-Transceivern die ISO Protokollschichten 1 und 2, d. h. die Bit- und Datenübertragungsschicht, während die sogenannten *MOST Network Services Layer 1 und 2* die Dienste der ISO-Schichten 3 bis 7 nahezu vollständig abdecken.

Der öffentlich zugängliche Teil der MOST Spezifikationen ist allerdings an einigen Stellen sowohl beim *Physical Layer,* der Datenübertragungsschicht als auch bei den höheren Schichten lückenhaft bzw. nur den Mitgliedern der *MOST Cooperation* zugänglich. Dieses teilweise nicht veröffentlichte und mit Patenten belegte Implementierungs-Know-how wahrt wohl kommerzielle Interessen, behindert aber die schnellere Verbreitung des Standards und sorgt für Verunsicherung, wenn wesentliche Systemkomponenten von einem einzigen Hersteller abhängig sind. Das Konsortium hat angekündigt, dass die Pioniere der MOST-Technologie, die über die entscheidenden Patentrechte verfügen, zukünftig eine offenere Lizenzpolitik betreiben wollen.

Gleichzeitig werden immer öfter Schnittstellen zu *Consumer*-Geräten wie MP3-Playern und Mobiltelefonen gefordert. Neben der Funktechnologie *Bluetooth* müssen USB-Schnittstellen zu Mobiltelefonen und MP3-Playern bereitgestellt und das Internet integriert werden (*Customer Convenience Port*). Daneben werden höhere Bitraten für neue Fahrer-Assistenzsysteme gefordert, wenn etwa Live-Videosignale von einer hochauflösenden Rückfahrkamera übertragen werden sollen. Im Gegensatz zu DVDs, bei denen die Bilder komprimiert gespeichert und übertragen werden, reicht die typische Rechenleistung eingebetteter Systeme in der Regel nicht aus, derartige Signale vor der Übertragung in Echtzeit zu komprimieren.

Nachdem eine Zeit lang überlegt wurde, das für den Automobileinsatz als *IDB 1394* schon standardisierte *Firewire* einzusetzen, gilt mittlerweile *Ethernet* als Favorit, weil damit praktisch alle Internet-tauglichen Geräte ins Fahrzeugnetz eingebunden werden können. Wenn aber *Ethernet* im Fahrzeug und für die Diagnoseschnittstelle ohnehin vorhanden ist, wäre es sinnvoll, wenn es MOST ersetzen und auch gleich die Aufgaben des Infotainment-Netzes übernehmen würde. Vereinzelt wird Ethernet im Forschungs- und Vorentwicklungsbereich sogar für den Echtzeiteinsatz, d. h. als Alternative zu FlexRay oder CAN, evaluiert (Abschn. 3.5). Die *MOST Cooperation* sieht sich hier in einem möglichen Verdrängungswettbewerb und versucht gegenzuhalten. Helfen sollen höhere Bandbreiten wie MOST150 mit 150 Mbit/s und zukünftig noch mehr, kostengünstigere Busverkabelungen und das Konzept, MOST als *Physical Layer* für *Ethernet*-Datenpakete einzusetzen.

Nicht vergessen werden darf in dieser Diskussion, dass MOST ein etabliertes, im Automobileinsatz ausgereiftes Bussystem ist. *Ethernet* hat sich zwar im Bürobereich seit vielen Jahren bewährt, für den Automobileinsatz mit seinen schwierigen Randbedingungen für den *Physical Layer* und seine Anforderungen an Echtzeitverhalten und Übertragungssicherheit muss es aber erst einmal fahrzeugtauglich gemacht werden.

### 3.4.1 Bus-Topologie und Physical Layer

Die erste und immer noch eingesetzte Generation der MOST Systeme (MOST25) basiert auf einem optischen Übertragungsmedium mit Kunststoff-Lichtwellenleitern (Plastic Optical Fiber POF). Das modulierte Lichtsignal überträgt die Daten Manchester-codiert, wodurch auf der Empfängerseite eine Bittaktsynchronisation möglich wird. Im Steuergerät wird der Lichtstrom von einem optischen Transmitter (*Fiber Optical Transmitter FOT*) empfangen bzw. wieder ausgesendet, zur Weiterverarbeitung im Kommunikationscontroller aber in elektrische Form umgewandelt. Neben der optischen gibt es mittlerweile auch eine elektrische Variante mit einer verdrillten Zwei-Draht-Leitung, die im Zusammenhang mit der zweiten Generation MOST50 eingeführt wurde. Als Weiterentwicklung steht MOST150 mit 150 Mbit/s wieder auf optischer Basis bereit. Zusätzlich gibt es für MOST150 auch eine elektrische Variante, die allerdings mit im Vergleich zu MOST50 teureren Koaxialkabeln arbeitet. Das MOST Übertragungsprotokoll selbst ist unabhängig vom physikalischen Übertragungsmedium.

MOST lässt unterschiedliche Bus-Topologien zu. Am häufigsten eingesetzt wird eine logische Ring-Struktur mit bis zu 64 Steuergeräten (Abb. 3.32). Physikalisch betrachtet sind die Verbindungen zwischen den einzelnen Steuergeräten unidirektionale Punkt-zu-Punkt-Verbindungen, d. h. das optische Signal wird in jedem Steuergerät regeneriert, bevor es zum nächsten Steuergerät weitergesendet wird. Jedes Steuergerät enthält deshalb genau einen Eingang und einen Ausgang. Inaktive Steuergeräte leiten die ankommenden Daten unverändert weiter (Bypass-Betrieb), aktive Geräte entnehmen Daten oder fügen eigene Daten in den Datenstrom ein. Ein vorher festgelegtes Gerät arbeitet als Master-Steuergerät (*Timing Master*) und erzeugt die Botschaften (*Frames*). Die übrigen Steuergeräte (*Slaves*) synchronisieren sich auf den Bit- und Frame-Takt des Masters und entnehmen aus den im Ring umlaufenden Botschaften Empfangsdaten bzw. fügen Sendedaten ein. Das Bussystem arbeitet in der Regel mit einer Bitrate von etwa 25 Mbit/s (MOST25), neuere Buscontroller auch mit 50 Mbit/s (MOST50) und 150 Mbit/s (MOST150). Die Durchlaufverzögerung eines Steuergerätes bei MOST25 beträgt 2 Frames, d. h. etwa 45 µs (siehe unten), bei MOST50 und MOST150 liegt sie unter 1 µs.

### 3.4.2 Data Link Layer

MOST verwendet eine bitstromorientierte Übertragung, bei der 16 Botschaften, hier Frames genannt, zu einem Block zusammengefasst werden (Abb. 3.33). Jede Botschaft durchläuft genau einmal den gesamten Ring. Die Botschaftsrate wird meist auf 44,1 kHz eingestellt und entspricht damit der Abtastfrequenz von Audio-CDs. Daraus ergibt sich eine Botschaftslänge von 22,67 µs. Alternativ sind 48 kHz möglich. Diese bei DVD-Audio Player oder DAT-Geräte übliche Rate wird seit der Spezifikation 3.0 empfohlen. Die Botschaftsrate muss für das gesamte Netz einheitlich sein. Weicht die interne Abtastrate eines Gerätes von der Botschaftsrate des Bussystems ab, muss das Gerät die Abtastrate konvertieren. MOST

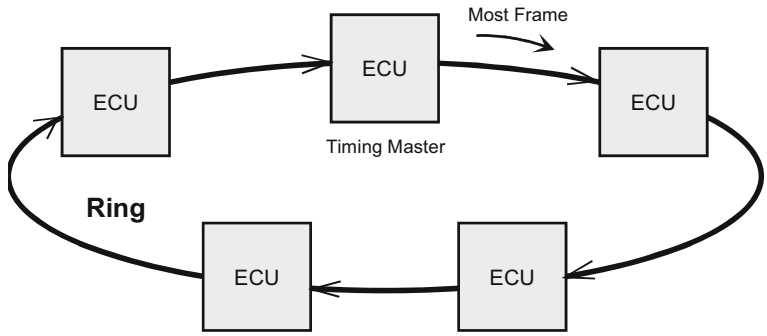


**Abb. 3.32** MOST-System in Ring-Struktur

ist grundsätzlich als synchrones System konzipiert, d. h. es wird in kontinuierlichen Taktzyklen immer die gleiche Anzahl an Bytes übertragen. Das Botschaftsformat wurde aber so flexibel gewählt, dass auch asynchrone Übertragungen innerhalb dieser synchronen Struktur möglich sind.

#### 3.4.2.1 MOST25 Frames

Jeder Frame beginnt mit dem Header, der sich aus einer *Präambel* (4 bit) und dem sogenannten *Boundary Descriptor* (4 Bit) zusammensetzt (Abb. 3.33). Die *Präambel* kennzeichnet den Start eines Frames und ermöglicht eine Neusynchronisation auf den ankommenden Bitstrom. Der Blockanfang wird durch eine abweichende *Präambel* im ersten der 16 Frames markiert. Der *Boundary Descriptor* unterteilt das nachfolgende Datenfeld in einen synchronen und einen asynchronen Bereich. Sein Wert bestimmt die Anzahl der synchronen Daten in Vielfachen von 4 Byte (*Quadlets*), wobei die Mindestlänge des synchronen Datenfeldes 24 Byte beträgt. Der Rest der insgesamt 60 Bytes des Datenfelds, d. h. maximal 36 Byte, wird für asynchrone Datenpakete verwendet. Der *Boundary Descriptor* ist für alle Frames gleich und wird im laufenden Betrieb nicht verändert. Auf das Feld mit den synchronen und asynchronen Daten folgt ein 2 Byte großes Feld für Steuerbotschaften. Das abschließende Trailer-Byte enthält weitere Steuer- und Statusinformationen und bietet eine Überprüfung auf Übertragungsfehler. Als Bitfehlerrate wird ein Wert von durchschnittlich $10^{-10}$ genannt.

#### 3.4.2.2 MOST50 Frames

MOST50 arbeitet mit derselben Frame-Rate und Block-Struktur wie MOST25, aber der doppelten Bitrate. Dadurch passen 128 Byte in einen Frame. Der Header umfasst nun 7 Byte, gefolgt von dem auf 4 Byte verdoppelten Feld für Steuerbotschaften, das in der Dokumentation meist als Teil des dann 11 Byte langen Headers dargestellt wird. Das jetzt 117 Byte lange Datenfeld kann über den *Boundary Descriptor* beliebig zwischen synchronen und asynchronen Daten aufgeteilt werden. Im Gegensatz zu MOST25, wo eine Änderung der Aufteilung nur durch Abbruch und Neuaufbau der synchronen Kommu-

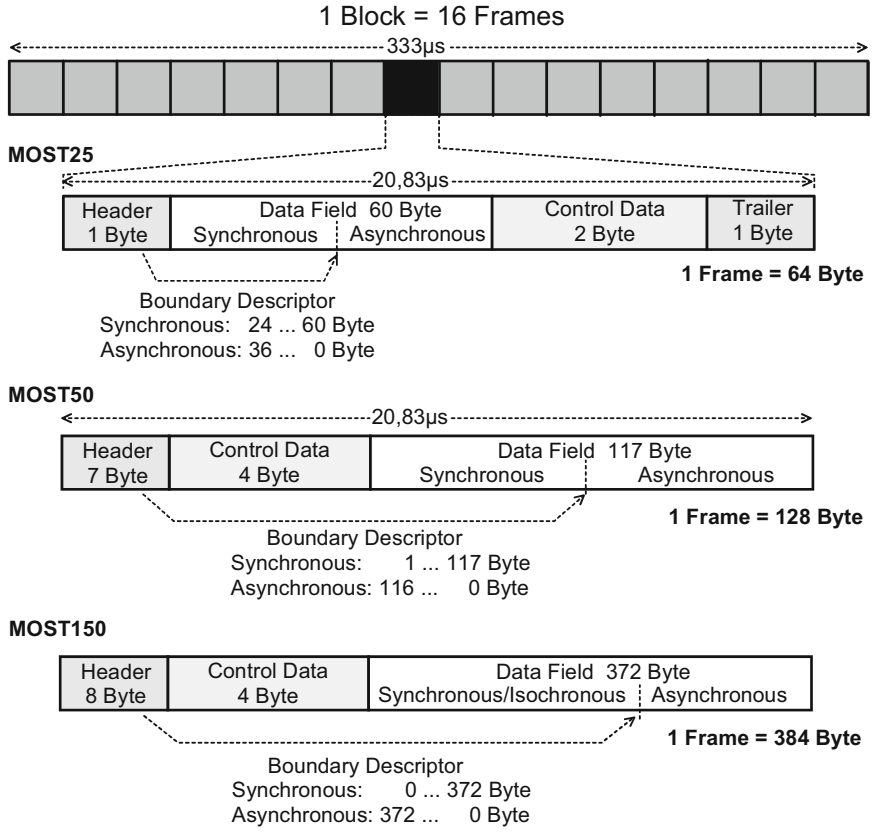


**Abb. 3.33** MOST-Botschaftsformat Block und Frame (Frame-Rate 48 kHz)

nikation möglich ist, lässt MOST50 ausdrücklich eine im laufenden Betrieb dynamisch veränderliche Aufteilung zu.

### 3.4.2.3 MOST150 Frames

MOST150-Frames sind, mit Ausnahme des um ein Byte längeren Headers, aufgebaut wie MOST50-Frames. Wegen des höheren Bittaktes bei gleicher Frame-Dauer passen bis zu 372 Datenbytes in einen Frame.

Entsprechend der Aufteilung der Datenfelder in einem Frame unterscheidet MOST zwischen drei voneinander unabhängigen Gruppen von Datenkanälen für unterschiedliche Anwendungszwecke:

- Synchrone Daten (*Streaming Data* synchron oder isochron), z. B. Audio, Video,
- Asynchrone Daten (*Packet Data*), z. B. TCP/IP Pakete,
- Steuerdaten (*Control Channel*), z. B. Netzmanagement, Gerätekommunikation.

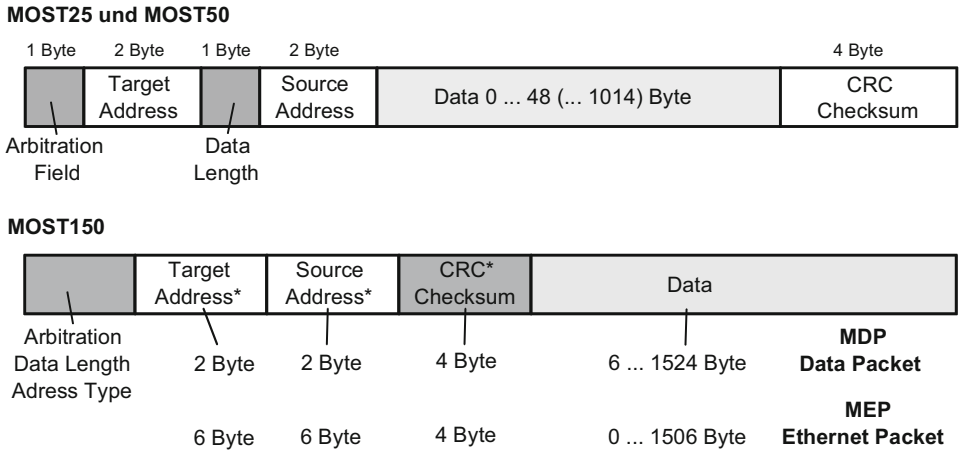


**Abb. 3.34** MOST-Datenpakete für asynchrone Daten (*Packet Data Channel PDC*)

Bevor auf diese drei Gruppen genauer eingegangen wird, ist allerdings eine Vorbemerkung zu dem hier häufig verwendeten deutschen Begriff *Botschaft* notwendig. Auf der Bitübertragungsschicht werden die drei Gruppen von Daten in einer gemeinsamen *physikalischen Botschaft* übertragen, in der MOST-Spezifikation mit dem englischen Begriff *Frame* bezeichnet. Innerhalb eines *Frames* besitzt jede Gruppe Datenfelder mit fester oder variabler Länge. Bei den asynchronen Daten bzw. den Steuerdaten sind diese Datenfelder nur Teile größerer *logischer Botschaften*, die in der Spezifikation als *Asynchronous Data Packet* (manchmal auch *Frame* statt *Packet*, Abb. 3.34) bzw. *Control Data Frame* (Abb. 3.35) bezeichnet werden. Im vorliegenden Text werden diese logischen Botschaften in der Regel als Datenpakete, gelegentlich aber auch einfach als Botschaften bezeichnet.

#### 3.4.2.4 Synchroner Datenbereich (Streaming Data)

Der *synchrone Datenbereich* wird in Zeitschlitze zu je 8 Bit eingeteilt, bei MOST als physikalische Kanäle bezeichnet. Mehrere dieser physikalischen Kanäle können dann zu einem logischen Kanal zusammengefasst werden. Ein einzelner logischer Kanal (*Streaming Channel*) kann dabei zwischen einem Byte und der Gesamtgröße des synchronen Datenbereichs verwenden. Der Buszugriff auf die Zeitschlitze erfolgt zeitsynchron, d. h. nach dem TDMA-Verfahren. Anwendungen fordern exklusive Kanäle an und können über diese Kanäle dann solange Daten übertragen, bis sie die Kanäle wieder freigeben. Das Format der Daten innerhalb dieser Kanäle ist beliebig. Bei Musiksignalen von Audio-CDs beispielsweise werden direkt die auf der CD gespeicherten 16 bit Abtastwerte übertragen. Für ein Stereo-Signal müssen dazu im synchronen Datenbereich vier physikalische Kanäle allokiert werden, jeweils zwei für das rechte und zwei für das linke Audio-Sample. Wird das gesamte Datenfeld ausgenützt (bei MOST25 60 Byte, bei MOST50 117 Byte, bei MOST150 372 Byte), können 15, 29 bzw. 93 unkomprimierte Audio-Stereo-CD-Kanäle übertragen werden. Synchrone

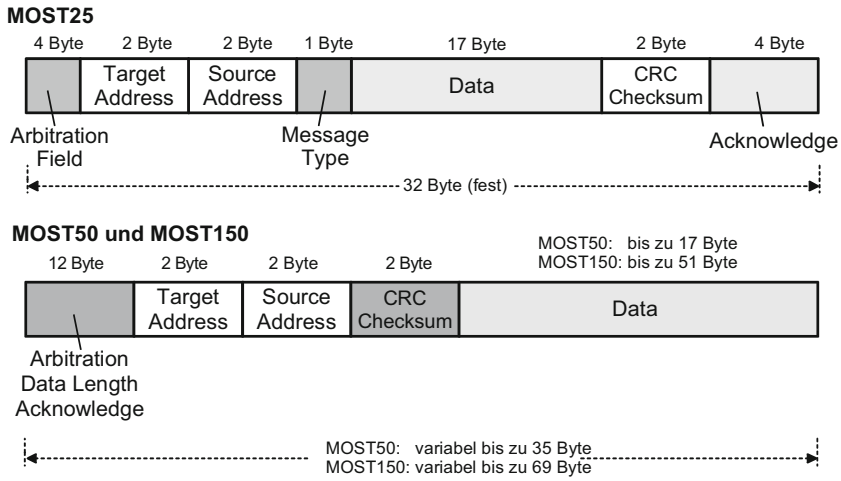


**Abb. 3.35** MOST-Datenpakete für Steuerdaten (*Control Data*)

Daten enthalten weder Sender- noch Empfängeradresse, die Administration erfolgt vollständig über die Steuerdaten (*Control Channel*). Genutzt wird der synchrone Datenbereich vor allem für Audio- und Videodaten, deren Übertragung eine hohe und garantierte Bandbreite benötigt.

Während einfache Audiodaten eine feste Abtastrate haben, die jedoch von der MOST Abtastrate abweichen kann, verwenden komprimierte Daten, z. B. MPEG-kodierte Videoströme, oft eine variable Datenrate, benötigen aber dennoch eine garantierte Bandbreite. Dazu müssen die Daten vor der Übertragung entweder auf die MOST- Abtastrate umkodiert werden oder die Frames müssen dynamisch mit Dummy-Daten aufgefüllt werden. Bei MOST25/50 musste dies von der Anwendung erledigt werden, bei MOST150 wurden zusätzlich sogenannte *Isochrone Kanäle* eingeführt, bei denen dies vom MOST Kommunikationscontroller (INIC) übernommen wird.

#### 3.4.2.5 Asynchroner Datenbereich (Packet Data)

Der *asynchrone Datenbereich* ist für formatierte Datenpakete vorgesehen (Abb. 3.34). Da je Frame nur eine begrenzte Zahl von Bytes für asynchrone Daten zur Verfügung steht, wird die Botschaft vom Kommunikationscontroller gegebenenfalls auf mehrere Frames aufgeteilt. Sender und Empfänger der Botschaft werden durch Geräteadressen identifiziert. Der Buszugriff erfolgt bei diesen Botschaften, sobald der asynchrone Datenbereich nicht durch ein anderes asynchrones Datenpaket belegt ist. Zur Erkennung dient das *Arbitration Field*. Details der Arbitrierung sind nicht offengelegt.

Standardmäßig können MOST25-Kommunikationscontroller in einem asynchronen Datenpaket maximal 48 Byte Nutzdaten versenden. Die Datenlänge wird im Feld *Data Length* in Vielfachen von 4 Byte (*Quadlet*) angegeben. Falls die Applikation, die den Kom-

munikationscontroller bedient, die Daten auf der Sendeseite schnell genug liefern bzw. auf der Empfangsseite schnell genug auslesen kann, lässt sich das Datenfeld bei MOST25 auf bis zu 1014 Byte erweitern. Bei MOST50-Systemen ist diese Länge der Standard. Asynchrone Botschaften werden im Kommunikationscontroller durch eine CRC-Prüfsumme gesichert. Die Spezifikation weist auf einen Bestätigungsmechanismus und eine automatische Sendewiederholung im Fehlerfall hin.

Bei MOST150 wurde das Format der Paketdaten verändert und zwei Formatvarianten eingeführt. Beim *MOST Data Packet MDP* Format wird weiterhin die bisherige 16 bit Geräteadressierung verwendet, es können aber jetzt bis zu 1524 Datenbyte übertragen werden. Das *MOST Ethernet Packet MEP* Format ist für das Durchleiten von Ethernet-Botschaften vorgesehen, ohne dass diese wesentlich umformatiert werden müssen. Daher werden für die Adressierung die bekannten 48 bit langen Ethernet MAC-Adressen und der bekannte Ethernet CRC statt der MOST-spezifischen Prüfsumme verwendet. Das Datenfeld kann in diesem Fall bis zu 1506 Byte übertragen.

Der asynchrone Datenbereich wird häufig für die Übertragung von Karteninformationen in Navigationssystemen oder für die Durchleitung von TCP/IP-Verbindungen eingesetzt, da hier kurzzeitig hohe Datenmengen anfallen können.

#### 3.4.2.6 Steuerdaten (Control Data)

Steuerdaten dienen zum einen der Verwaltung des Netzwerkes, zum anderen aber auch der Kommunikation zwischen den Anwendungen in den verschiedenen Steuergeräten im Ring. Die Übertragung findet ereignisorientiert im sogenannten *Control Channel (Kontrollkanal)* statt und bietet nur eine geringe Bandbreite. Die Steuerbotschaft hat bei MOST25 eine feste Länge von 32 Byte (Abb. 3.35). Diese 32 Byte werden beginnend mit dem *Arbitration Field* in jeweils 2 Byte große Einheiten zerstückelt und auf die *Control Data* Felder der 16 aufeinander folgenden Frames eines Blocks verteilt (Abb. 3.33). Die ersten beiden *Arbitration* Bytes der Steuerbotschaft nach Abb. 3.35 werden also im *Control Data* Feld des ersten Frames eines Blocks nach Abb. 3.33, die weiteren zwei *Arbitration Bytes* im *Control Data* Feld des zweiten Frames, die *Target Address* im *Control Data* Feld des dritten Frames übertragen usw. Die Definition eines Blocks ist nur für den Kontrollkanal relevant, da bei MOST25 pro Block genau eine Steuerbotschaft übertragen wird. Die synchronen und asynchronen Datenkanäle dagegen ignorieren Blockgrenzen.

Der Buszugriff auf den Kontrollkanal erfolgt asynchron und prioritätsbasiert, d. h. nach einem CSMA-Verfahren, wobei im Arbitrierungsfeld die Priorität der Botschaft angegeben wird. Steuerbotschaften werden durch eine Prüfsumme und eine Bestätigungsmechanismus geschützt. Im Fehlerfall kommt es zur automatischen Sendewiederholung (*Low Level Retries*). Eine erfolgreiche Übertragung wird vom Empfänger durch eine positive Bestätigung (*Acknowledge*, kurz *ACK*) quittiert. Findet sich kein Abnehmer für eine Botschaft, z. B. durch falsche Adressierung oder wenn der Empfänger keine Puffer zur Zwischenspeicherung frei hat, so interpretiert der Sender dies als ein *Not Acknowledge*, kurz *NAK*.

Im *Target* bzw. *Source Address* Feld wird eine 16 Bit lange Empfänger- bzw. Senderadresse angegeben. Ein einzelnes Steuergerät kann über seine physikalische oder logische

Adresse angesprochen werden. Die physikalische Adresse berechnet sich aus der relativen Position des Steuergerätes im Ring bezogen auf das Master-Steuergerät und wird meist nur beim Systemstart verwendet. Das Master-Steuergerät besitzt stets die physikalische Adresse $400_H$, alle nachfolgenden Slave-Steuergeräte erhalten darauf basierend eine um jeweils eins erhöhte Adresse. Die logische Adresse kann individuell vom *Network Master* pro Teilnehmer vergeben werden, muss aber wie die physikalische Adresse ebenfalls im Ring eindeutig sein. Üblicherweise beginnt man hier mit $100_H$ für den *Network Master* und inkrementiert dann ebenfalls um eins für jedes folgende Gerät. Dieses Verhalten ist jedoch nicht standardisiert. Außerdem besteht die Möglichkeit, eine Steuerbotschaft an eine Gruppe von zusammengehörenden Geräten (*Groupcast*, typisch *Target Address* $300_H$+*FBlockID*) oder an alle Geräte im Ring (*Broadcast*, typisch *Target Address* $3C8_H$) zu schicken.

Das Feld *Message Type* charakterisiert die Art der Steuerbotschaften. Neben normalen Botschaften, deren Bedeutung auf Anwendungsebene definiert wird, stehen vordefinierte Steuerbotschaften für die Netzwerkverwaltung zur Verfügung. Die *Resource Allocation* und *Resource Deallocation* Botschaften beispielsweise dienen der Reservierung und Freigabe von synchronen Übertragungskanälen. Mit *Remote Read* und *Remote Write* Botschaften kann ein Kommunikationscontroller, z. B. derjenige des Master-Steuergerätes, die Register und damit die Konfiguration eines anderen Kommunikationscontrollers auslesen oder verändern. Über *Remote Get Source* Botschaften kann abgefragt werden, welches Steuergerät Daten in einem bestimmten synchronen Datenkanal sendet.

Die Formatierung der Steuerbotschaften erfolgt selbstständig durch den MOST Kommunikationscontroller. Dieser überprüft auch die Empfängeradresse *Target Address* und die *CRC Checksum* auf Richtigkeit, bevor die Nutzdaten der Steuerbotschaft an die nächst höhere Schicht des Empfängers weitergereicht werden.

Bei MOST50 und MOST150 werden je Frame 4 Byte der Steuerbotschaft übertragen (Abb. 3.35). Das Format der Steuerbotschaft hat sich gegenüber MOST25 geändert. Ihre Länge ist jetzt variabel mit bis zu 35 bzw. 69 Byte, die auf mehrere Frames verteilt werden. Blockgrenzen spielen bei MOST50 und MOST150 keine Rolle mehr.

#### 3.4.2.7 Arbitrierung und Bandbreite

Während im synchronen Datenbereich jeder Sender die notwendige Übertragungsbandbreite fest für sich reservieren kann, konkurrieren die verschiedenen Steuergeräte um die Bandbreite bei den asynchronen und den Steuerdaten. Bei den asynchronen Botschaften wird die Sendeberechtigung mittels des Arbitrierungsfelds von einem Gerät zum anderen weitergereicht (*Token Passing*).

Bei den Steuerdaten erfolgt der Buszugriff über einen, wie die MOST-Spezifikation es ausdrückt, *fairen CSMA-Arbitrationsmechanismus*. Dabei spielt die Priorität der Botschaften eine Rolle, die zwischen 0 (nieder) und 15 (hoch) gewählt werden kann. Aufgrund der geringen Anzahl von Prioritätsstufen sind Konflikte möglich, zumal Steuerbotschaften bei gängigen Kommunikationscontrollern oft mit der Default-Priorität 1 versendet werden. Wie das Token Passing oder die Arbitrierung im Detail erfolgen und welche Worst Case *Latenzzeiten* sich dabei ergeben, lässt der öffentlich zugängliche Teil der Spezifikation

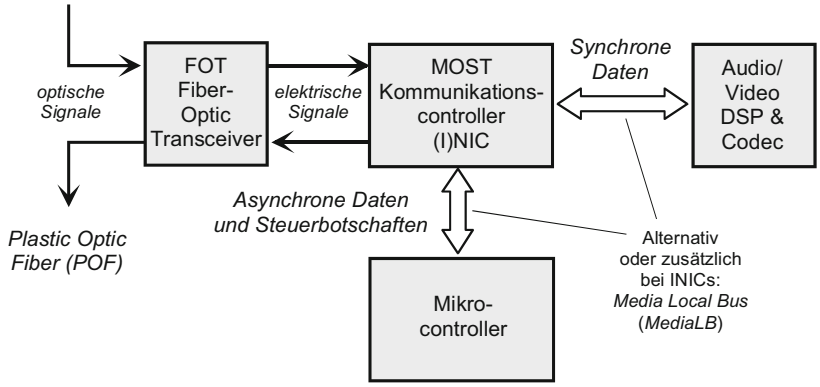


**Abb. 3.36** Typischer Aufbau eines MOST-Steuergerätes

allerdings weitgehend im Dunklen. Spezifiziert ist lediglich, dass die Steuerbotschaft mit der höchsten Priorität *gewinnt* und ein einzelner Kommunikationscontroller beim Senden nicht in jedem Frame senden darf.

Die Bestimmung der Bandbreite für die Übertragung ist relativ komplex. Neben der Bitrate, die die Länge der einzelnen Botschaften beeinflusst, sowie der Abtastrate spielt die Aufteilung in synchrone und asynchrone Kanäle eine wichtige Rolle. Ausführliche Beispielberechnungen finden sich in [17].

### 3.4.3 Kommunikationscontroller

Bei MOST handelt es sich um eine Technologie, die spezifisch für Infotainment-Anwendungen ausgelegt ist und immer noch hauptsächlich in den oberen und mittleren Fahrzeugklassen eingesetzt wird. Dieser geringe Verbreitungsgrad spiegelt sich direkt in einem begrenzten Angebot der Halbleiterhersteller für Kommunikationsbausteine wieder. Während bei CAN der Kommunikationscontroller typischerweise als On-Chip-Modul in vielen Mikrocontrollern bereits integriert ist, ist der Kommunikationscontroller bei MOST meist ein externer Baustein, der über verschiedene Schnittstellen mit anderen Prozessoren im Steuergerät kommuniziert (Abb. 3.36). Aufgrund der unterschiedlichen Frame-Formate sind dabei MOST25, MOST50 und MOST150 Kommunikationscontroller nicht abwärts kompatibel.

Die ersten verfügbaren MOST Kommunikationscontroller, auch als *Network Interface Controller NIC* bezeichnet, implementierten den kompletten Data Link Layer und kanalisierten die Datenströme zu den entsprechenden Datenquellen und Datensenken. In der Regel werden der asynchrone Datenkanal und der Kontrollkanal zusammen über eine Schnittstelle (Parallelbus oder $I^{2C}$) an den steuernden Mikrocontroller des Gerätes übertragen. Die geforderte Bandbreite hierbei ist eher gering. Die Audio- und Videoinhalte in

den synchronen Datenströmen dagegen benötigen eine hohe Bandbreite und werden über eine separate Schnittstelle, als $I^2S$ oder *Streaming Port* bezeichnet, an spezielle Chips zur Signalverarbeitung z. B. MPEG Dekoder, übertragen.

Der gesamte MOST Datenstrom wird vom Kommunikationscontroller empfangen und danach mit einer kurzen Verzögerung auf dem Ring weitergegeben. Während dieser Verzögerung fügt der Kommunikationscontroller zu sendende Daten in den empfangenen Frame ein bzw. extrahiert Daten, die für das lokale Steuergerät bestimmt sind. Für den Kontrollkanal und die asynchronen Daten entscheidet die Zieladresse der jeweiligen Botschaft. Für die synchronen Daten werden in einer *Routing Tabelle* die Kanäle und Zeitschlitze definiert, in welchen empfangen oder gesendet werden soll.

Der Datenaustausch zwischen der Infotainment-Anwendung auf dem Mikrocontroller und dem Kommunikationscontroller wird von den sogenannten *Funktionsblöcken* und *Network Services* gekapselt. Diese waren ursprünglich als reine Softwareschicht im Mikrocontroller konzipiert, während der Kommunikationscontroller selbst eine klassische Schnittstelle mit Steuer- und Datenregistern hatte. Die *Network Services* stellen eine Schnittstelle zur Verfügung, die direkt auf dem MOST-Nachrichtenformat mit *FBlockID*, *FktIDs* usw. (siehe unten) basiert. Die Einstellparameter des Kommunikationscontrollers werden ebenfalls über spezielle Funktionsblöcke zugänglich gemacht.

Neuere Generationen von Kommunikationscontrollern, sogenannte *Intelligent Network Interface Controller INIC*, enthalten bereits die Funktion des *Network Service Layer* und entlasten damit den Mikrocontroller. Dadurch wird auch der Übergang von MOST25 zu den neueren Generationen erleichtert, weil die INIC API unverändert bleiben soll. Eine weitere Neuerung der *INIC*-Generation ist die Einführung des *Media Local Bus* (*MediaLB*), der die traditionellen Schnittstellen zur Anbindung des Mikrocontrollers, DSPs, Codecs usw. ablösen und die volle MOST50/150-Bandbreite auch steuergeräteintern unterstützt.

### 3.4.4 Network Services und Funktionsblöcke

Anders als die bisher vorgestellten Bussysteme, bei denen die höheren Protokollschichten in der Regel gar nicht oder erst viel später als der Physical und der Data Link Layer und oft nicht durchgängig festgelegt wurden, spezifiziert MOST zusätzlich Mechanismen auf höherer Ebene. Die Schnittstelle zu diesen höheren Schichten wird von den sogenannten *MOST Network Services* (kurz *NetServices*) gebildet, die von MOST in *Layer 1* und *Layer 2* unterteilt werden. Grob kann man *MOST Layer 1* den Schichten 3 bis 5 des ISO/OSI-Schichtenmodells und *Layer 2* der Schicht 6 zuordnen (Abb. 3.37). Auf den *Network Services* setzen die *Funktionsblöcke* auf, die eine objektorientierte Programmierschnittstelle für die Infotainment-Anwendungen darstellen und der ISO/OSI-Schicht 7 zuzuordnen sind.

Die MOST-Spezifikation definiert die *Network Services* als eine Reihe von Funktionen, mit denen die Zugriffe auf den Kommunikationscontroller und das Netz durchgeführt werden. Einige Hersteller bieten zu ihren Kommunikationscontrollern eine passende Software-Implementierung dieser Dienste an. Neuere MOST Kommunikationscontroller

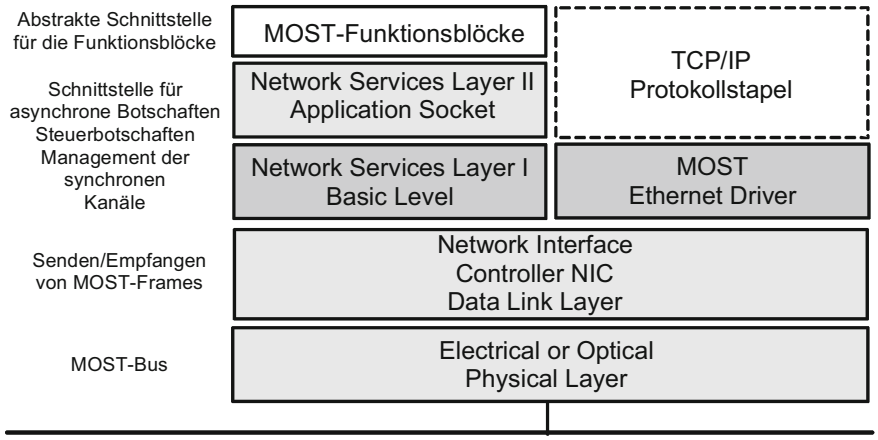


**Abb. 3.37** MOST-Protokollstapel mit Erweiterung für Ethernet bei MOST150

(*INIC*) enthalten bereits eine Hardware-Implementierung der *Layer I*-Dienste. Die *Network Services* verwalten alle drei Datenkanalgruppen. Für die asynchronen Daten und die Steuerbotschaften stellen die *Network Services* logische Protokollebenen bereit, während synchrone Datenkanäle lediglich verwaltet und die Nutzdaten an die anwendungsspezifische Signalverarbeitung weiterleitet werden.

Die *Layer I*-Dienste umfassen folgende Funktionsgruppen:

- *Synchronous Channel Service SCS* bzw. *Socket Connection Manager SCM*: Reservierung und Freigabe synchroner Kanäle, Konfiguration der Schnittstelle für synchrone Daten (siehe Abb. 3.36). Neuere Netzwerkcontroller (INIC) kapseln das Management synchroner Botschaften durch sogenannte *Sockets*, während bei älteren Kommunikationscontrollern (NIC) die Reservierung und Freigabe von Kanälen und die Verbindungen zur Signalquelle (*Source*) und Signalsenke (*Sink*) explizit durchgeführt werden mussten (siehe Abschn. 3.4.7).
- *Asynchronous Data Transmission Service ADS*: Senden und Empfangen von asynchronen Daten inklusive Zwischenspeicherung und Fehlerbehandlung.
- *Control Message Service CMS*: Senden und Empfangen von Steuerdaten.
- *Application Message Service AMS*: Senden und Empfangen über den Kanal für Steuerdaten mit Segmentierung, falls mehr als 17 Byte Nutzdaten notwendig sind.
- *MOST Transceiver* bzw. *Processor Control Service MCS*: Konfiguration des Kommunikationscontrollers, Setzen des *Boundary Descriptors* und der Frame-Rate.
- *MOST Supervisor MSV*: Konfiguration als *Timing Master* oder *Slave*, Power Management, Netzwerk Start und Stopp (siehe Abschn. 3.4.5).

Zu den *Layer II*-Diensten gehören:

- *Command Interpreter CMD*: Verarbeitung von Botschaften mit Steuerdaten und Aufruf von Anwenderfunktionen für deren Bearbeitung.
- *Address Handler AH* und *Network Master Shadow*: Zugriff auf die Kopie von Datenstrukturen des *Network Masters* wie der *Central Registry* (siehe Abschn. 3.4.5) zur Verwaltung des Netzwerks.
- *Notification Services NTFS*: Automatischer Benachrichtigungsmechanismus, wenn sich Parameter (*Properties*) eines durch einen Funktionsblocks verwalteten Gerätes ändern (siehe unten).

Über den *Layer II*-Diensten liegen die sogenannten Funktionsblöcke, die wie alle übrigen MOST-Objekte mit Hilfe von Kennziffern referenziert werden. Bei den Funktionsblöcken wird diese Kennziffer als *FblockID* bezeichnet. Da einige Funktionsblöcke, z. B. die Diagnose, mehrfach in einem System vorkommen können, muss zur eindeutigen Kennzeichnung zusätzlich zur Funktionsblock-Kennziffer auch eine Instanznummer *InstID* angegeben werden. Die Gruppe der Audio-Funktionsblöcke beispielsweise umfasst unter anderem den Audio-Verstärker (*FBlockID*=22h), die Freisprecheinrichtung (*Handsfree Processor FBlockID*=28h) oder die Audiosignalverarbeitung mit Lautstärkecontroller, Equalizer usw. (*Audio DSP FBlockID*=21h), während das Bedienpanel zur Funktionsgruppe *Human Machine Interface* (*FBlockID*=10h) gehört. Jedes Steuergerät bietet eine bestimmte Anzahl an Funktionsblöcken für den jeweiligen Aufgabenbereich an. Die eigentliche Infotainment-Funktion, beispielsweise das Abspielen einer CD, wird durch das Zusammenspiel mehrerer solcher Funktionsblöcke realisiert, wobei es für die Anwendung gleichgültig ist, ob die Funktionsblöcke im selben Gerät enthalten oder über das Netz verteilt sind, da die Kommunikation zwischen den Funktionsgruppen über die *Network Services* gekapselt wird.

Neben den für die eigentliche Infotainment-Anwendung wichtigen Funktionsblöcken muss jedes Gerät den Funktionsblock *NetBlock* (*FBlockID*=01h), das Master-Steuergerät die Blöcke *NetworkMaster* bzw. *ConnectionMaster* (*FBlockID*=02h und 03h) implementieren, in denen der Zugriff auf den Bus und das Netzmanagement enthalten ist. Weiterhin kann man davon ausgehen, dass der Funktionsblock *Diagnosis* (*FBlockID*=06h) in jedem Steuergerät zu finden ist, über den häufig auch höhere Protokolle wie KWP 2000 oder UDS implementiert werden. Für diejenigen Steuerbotschaften, die zur Koordination des Busbetriebs zwischen den Kommunikationscontrollern versendet werden, existiert zusätzlich der Pseudoblock *GeneralFBlock* mit *FBlockID*=00h. Neben den von MOST definierten Funktionsblöcken gibt es auch einen Bereich, den die Fahrzeughersteller für proprietäre Zwecke nutzen können, z. B. zur Systemüberwachung oder für den Software-Download.

Infotainment-Anwendung sind typischerweise über mehrere Steuergeräte verteilt, z. B. Bediengerät, Radioempfänger, Vorverstärker und aktive Lautsprecher, die über MOST vernetzt sind. Die örtliche Verteilung der Funktionsblöcke und die zwischen ihnen und der Anwendung notwendige Kommunikation über das Bussystem werden durch die *Net-*

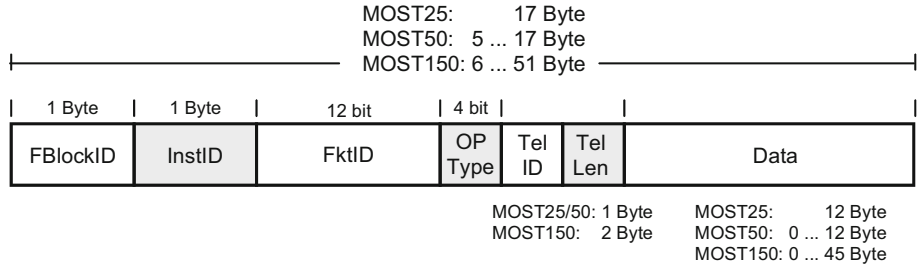


**Abb. 3.38** Nutzdatenbereich einer Steuerbotschaft für eine *Network Service* Funktion

*work Services* gekapselt. Diese Anwendungen rufen die Funktionen einfach mit Hilfe von *FBlockID* und *FktID* auf, unabhängig davon, in welchem Gerät diese Funktionen implementiert sind. Die Auflösung der Netzwerkadressen sowie das Versenden der notwendigen Busbotschaften werden von den *Network Services* übernommen. Dazu werden die Funktionen und ihre Parameter direkt auf den Nutzdatenbereich der Steuerbotschaften abgebildet (Abb. 3.38).

Jeder Funktionsblock stellt eine Reihe von Funktionen (*FktID*) bereit. MOST unterscheidet dabei zwischen *Properties* und *Methods*. *Properties* sind Zustände oder Eigenschaften, deren aktueller Wert durch eine entsprechende *Network Service* Funktion abgefragt oder unmittelbar, d. h. ohne wesentliche zeitliche Verzögerung, verändert werden kann. Ein typisches Beispiel wäre die Lautstärke eines Audioverstärkers. *Methods* sind Aktionen, die durch eine entsprechende *Network Service* Funktion angestoßen werden, dann aber längere Zeit für ihre Ausführung benötigen, bevor ein Ergebnis vorliegt, etwa der Sendersuchlauf des Radioempfängers. Detaillierte Angaben, wie die Funktion ausgeführt werden soll, erfolgen über *OPType*. Typisch für *Properties* sind Operationen wie `Set()` oder `Get()`, bei *Methods* Operationen wie `Start()` oder `Abort()`. Als Parameter für die Funktionen steht eine optionale Parameterliste zur Verfügung, deren variable Länge über das Feld *TelLen* angegeben wird. Falls mehr als 12 Byte benötigt werden, werden die Daten einer logischen Steuerbotschaft auf mehrere physikalische Steuerbotschaften aufgeteilt (segmentiert). Das Feld *TelID* dient dann dazu, die Segmentnummer zu übertragen, bei unsegmentierten Botschaften ist es 0. Die *Network Services* übernehmen sowohl die Segmentierung auf der Senderseite als auch das Zusammensetzen auf der Empfängerseite. Auf Applikationsebene können damit Botschaften mit bis zu 64 KB Daten transparent übertragen werden (*Application Message Service AMS*).

Gewöhnungsbedürftig im Umgang mit den Funktionsblöcken ist, dass die Funktionen bzw. zugehörigen Steuerbotschaften oft als Bündel der verschiedenen Kennziffern, allgemein *FBlockID.InstID.FktID.OpType(Parameter)* dokumentiert werden. Beispielsweise steht *22.00.400.0.(20)* für *AudioAmplifier.00.Volume.Set(20)*, d. h. der Audioverstärker (*FBlockID*=22h) mit der Instanznummer *InstID*=00h, d. h. der erste vorhandene Verstärker, erhält die Anweisung, die Lautstärke (*FktID*=400h) neu einzustellen (*OpType*= 0), wobei der gewünschte Wert 20 (auf einer Skala von 0 … 255) ist.

Die hier aufgezeigte Struktur für Steuerbotschaften (*FBlockID*, *FktID* usw.) wird prinzipiell auch für den asynchronen Datenkanal verwendet. Einziger Unterschied ist die Größe der *TelLen*, welche dort auf 12 bit erweitert wurde, um damit eine maximal zulässige Nutzdatenmenge von bis zu 1008 Byte zu spezifizieren.

### 3.4.5 Netzmanagement

MOST kennt verschiedene Rollen zur Verwaltung des Netzwerkes, die theoretisch auf verschiedene Geräte verteilt werden könnten, praktisch aber meist innerhalb der zentralen Bedieneinheit für das Infotainmentsystem im Armaturenbrett, der *Head Unit*, zusammengefasst werden:

- *Timing Master*: Erzeugung des Bit- und Frame-Taktes.
- *Network Master*: Führt Buch über die Systemkonfiguration, d. h. die im Ring vorhandenen Geräte und deren Eigenschaften (*Central Registry*).
- *Power Master*: Koordiniert das Herunterfahren (*Shutdown*) des Bussystems.
- *Connection Master*: Verwaltet die synchronen Übertragungskanäle.

#### 3.4.5.1 Start und Stopp des Bussystems

Die Verwaltung des Netzes erfolgt durch eine Mischung von Hardware innerhalb des Kommunikationscontrollers und Software im steuernden Mikrocontroller. Der *Network Master* durchsucht beim Start das Netzwerk nach angeschlossenen Steuergeräten (*System Scan*), fragt deren Eigenschaften ab und legt eine Tabelle (*Central Registry*) mit diesen Informationen an. Bei dieser Abfrage wird u. a. ermittelt, welche Geräte welche Funktionsblöcke enthalten. Falls die Steuergeräte nicht neu eingeschaltet, sondern lediglich in einem energiesparenden Zustand (*Sleep*) waren, werden sie durch das Lichtsignal bzw. elektrische Empfangssignal aufgeweckt. Das Abschalten des Bussystems wird vom *Power Master* über Statusbotschaften (*NetBlock.InstID.Shutdown.Query*) angekündigt, danach hat jedes Steuergerät Zeit, die notwendigen Vorbereitungen zu treffen, bevor das Master-Steuergerät das Sendesignal abschaltet. Die Slave-Geräte reagieren auf die ausbleibenden Signale und gehen, zumindest an der Busschnittstelle, ebenfalls in den *Sleep* Zustand über. Das Aufwecken des Bussystems aus dem *Sleep* Zustand kann durch jedes Gerät am Bus erfolgen, in dem es sein Sendesignal wieder einschaltet. Dadurch wachen der Reihe nach alle Geräte im Ring auf und der *Timing Master* beginnt mit dem Aussenden von Frames. Sobald sich die Geräte auf den Takt der Empfangssignale synchronisiert haben (*Lock*, siehe unten), beginnt der *Network Master* mit dem beschriebenen Durchsuchen des Netzes.

#### 3.4.5.2 Systemzustände

Aus Sicht der Anwendung befindet sich der MOST Bus in einem der beiden Zustände *OK* oder *Not OK*. Der momentane Systemzustand wird vom *Network Master* bestimmt und bei

Änderungen über die Nachricht *NetworkMaster.InstID.Configuration.Status* an alle Busteilnehmer verteilt. Bei jedem Systemstart wird zunächst vom Zustand *Not OK* ausgegangen. Sobald der *Network Master* die *Central Registry* erfolgreich aufgebaut hat, wird der Zustand *OK* eingenommen und die Steuergeräte fangen an, miteinander zu kommunizieren. Ändert ein Steuergerät seine Eigenschaften oder stellt der *Network Master* eine Änderung bzw. einen Fehler fest, z. B. ein ab- oder neu eingeschaltetes Gerät, so wird dies allen Steuergeräten über obige Statusbotschaft als *Not OK* mitgeteilt. Dadurch werden alle synchronen Verbindungen unterbrochen und durch den Neuaufbau der *Central Registry* das Netz rekonfiguriert. Bei weniger kritischen Änderungen verbleibt das Netz im Zustand *OK* und es wird lediglich eine Nachricht mit den entsprechenden Informationen über die Änderung verschickt.

Neben diesen beiden Zuständen, die das gesamte System betreffen, verwalten die *Network Services* noch weiteren Zustände individuell für jedes Steuergerät. Die wichtigsten sind *Lock* und *Unlock*. Detektiert werden sie vom Kommunikationscontroller, haben aber direkte Auswirkungen auf die Applikation. Der Zustand *Lock* wird vom Steuergerät dann eingenommen, wenn es sich auf die eingehenden Lichtimpulse synchronisieren kann. Kommt es zu einer Störung dieser Synchronisation, meldet der Kommunikationscontroller dem steuernden Mikrocontroller einen *Unlock*, da dann von fehlerhaft empfangenen Daten auszugehen ist. Die Anwendung wird darauf reagieren und im Fall eines Verstärkers etwa das Audiosignal stumm schalten.

#### 3.4.5.3 Verwaltung der synchronen Kanäle

Wenn ein Gerät einen Übertragungskanal im synchronen Datenbereich anfordert oder wieder freigibt, reserviert dieses Gerät den Kanal über eine *Resource Allocation* bzw. *Resource Deallocation* Botschaft und erhält eine zustimmende oder ablehnende Bestätigung. Der *Connection Master* trägt die Reservierung in eine lokale Tabelle ein (*Resource Allocation Table*), die automatisch (alle 1024 Frames) an die Slave-Steuergeräte verteilt wird. Die Anwendungssoftware muss auf der Netzwerkebene lediglich eingreifen, wenn die geforderte Übertragungsbandbreite nicht mehr frei ist.

#### 3.4.5.4 Ringbruch-Diagnose für die optischen Verbindungen

Bei einer ringförmigen Topologie führt eine einzelne Unterbrechung zum Totalausfall des Gesamtsystems. Mögliche Szenarien, die zu einer Unterbrechung des Rings führen, sind ein defektes Steuergerät oder ein echter Bruch eines Lichtwellenleiters. MOST spezifiziert innerhalb der *Network Services* nur wenige Diagnosemöglichkeiten, wie etwa die beschriebenen *Lock* bzw. *Unlock* Situationen. Die Fahrzeug- bzw. Gerätehersteller verwenden daher proprietäre Diagnosestrategien, die aber praktisch alle auf den gleichen Kernprinzipien beruhen. Die MOST Steuergeräte erhalten neben der optischen Schnittstelle zusätzlich eine elektrische Ein-Draht-, LIN- oder CAN-Schnittstelle. Über diese wird das Gerät geweckt und, wenn auch eventuell sehr primitiv, mit dem Gerät bidirektional kommuniziert. Im Fall der Ringbruch-Diagnose werden alle Geräte aufgefordert, Lichtsignale auszugeben. Danach wird von jedem Steuergerät einzeln abgefragt, ob es selbst ein Lichtsignal empfängt.

Damit lässt sich herausfinden, zwischen welchen Geräten im Ring sich die Fehlerquelle befindet.

Bei elektrischen statt optischen Verbindungen sind alternative Arten der Ringbruchdiagnose einsetzbar, falls erforderlich.

### 3.4.6 Höhere Protokollschichten

Parallel zu den Funktionsblöcken existiert das in Software zu implementierende Übertragungsprotokoll *MOST High Protocol MHP*. Es ist für die verbindungsorientierte Übertragung zwischen zwei MOST-Geräten vorgesehen und erlaubt die segmentierte Übertragung größerer Datenmengen mit Verbindungsaufbau und -abbau sowie Flusssteuerung. MHP setzt auf den *Network Services Level I* auf und verwendet wahlweise den asynchronen Datenbereich oder Datenpakete für Steuerdaten. Alternativ zu *MHP* können mit Hilfe des *MOST Asynchronous Media Access Control MAMAC* Protokolls unmittelbar Ethernet-Botschaften über MOST25 übertragen werden. Mit *MHP* und *MAMAC* können MOST25-Systeme mit TCP/IP basierten Notebooks oder dem Internet gekoppelt bzw. zur Weiterleitung derartiger Botschaften verwendet werden. Bei MOST150 ist *MAMAC* nicht mehr nötig, da die Unterstützung von *MOST Ethernet Packets MEP* Botschaften direkt in den asynchronen Datenkanal integriert ist.

Audio- und Videodaten sind aus Kopierschutzgründen oft verschlüsselt abgelegt. Da die Nutzungsbedingungen die unverschlüsselte Übertragung über ein aus der Sicht der Musik- und Filmindustrie vermeintlich offenes Datennetz wie MOST verbieten, muss neuerdings auch eine verschlüsselte Datenübertragung bereitgestellt werden, bei der Sender und Empfänger sich authentifizieren und Schlüssel austauschen. Dazu wird *Digital Transmission Content Protection DTCP* eingesetzt, ein Verfahren das aus der Unterhaltungselektronik stammt und auch bei IEEE 1394-FireWire verwendet wird.

### 3.4.7 Beispiel für Systemstart und Audioverbindung

In Abb. 3.39 ist beispielhaft der Aufbau eines MOST25 Rings mit vier Teilnehmern dargestellt. Der *Network*, der *Connection* und der *Timing Master* werden in der Regel im zentralen Infotainment-Steuergerät implementiert, der sogenannten *Head Unit,* die auch als Schnittstelle zum Benutzer des Systems dient. Die anderen drei Teilnehmer (*Slaves*) stellen Dienste bereit, die von der *Head Unit* genutzt werden. Das Beispielnetz besteht aus einem Navigationssystem, einem CD-Spieler und einem Verstärker.

Die in Tab. 3.13 dargestellte Aufzeichnung von Steuerbotschaften zeigt den Startvorgang des Systems sowie den Aufbau einer synchronen Audioverbindung zwischen CD-Player und Audioverstärker in leicht vereinfachter Form.

**Tab. 3.13** Steuerbotschaften beim Startvorgang des MOST-Rings nach Abb. 3.39

| Von | Nach | Msg. Type | Steuerbotschaften |
|---|---|---|---|
| 0100 | 0400 | 04 | **ResourceDeAllocate.DeallocateAll**<br>Beim Start eines MOST Netzwerkes setzt der *Network Master* die Belegung der synchronen Kanäle zurück und versendet die Systemnachricht *Resource Deallocation*. Systemnachrichten werden stets über den Pseudo-Funktionsblock 00h mit Zieladresse 400h verschickt und von den *Network Services* verarbeitet. Auf Applikationsebene ist diese Nachricht nicht sichtbar. |
| 0100 | 0401 | 00 | **NetBlock.01.FBlockIDs.Get** |
| 0101 | 0100 | 00 | **NetBlock.01.FBlockIDs.Status. 520053000600** |
| 0100 | 0402 | 00 | **NetBlock.02.FBlockIDs.Get** |
| 0102 | 0100 | 00 | **NetBlock.02.FBlockIDs.Status. 31000600** |
| 0100 | 0403 | 00 | **NetBlock.03.FBlockIDs.Get** |
| 0103 | 0100 | 00 | **NetBlock.03.FBlockIDs.Status. 22000600**<br>Diese Folge von Nachrichten stellt den sogenannten *System Scan* dar, bei dem der *Network Master* die *Slaves* der Reihe nach abfragt, um die unterstützten Funktionsblöcke zu ermitteln. Dabei wird die physikalische Adresse (400h + Position im Ring) verwendet, da die logische Adresse zu diesem Zeitpunkt noch undefiniert ist. Die Navigation meldet *FblockID*=52h (*Navigation System*), 53h (*TMC Decoder*) und 06h (*Diagnosis*), jeweils mit *InstID*=00h. Der CD-Spieler unterstützt die Funktionsblöcke 31h (*Audio Disk Player*) und 06h (*Diagnosis*). Der Verstärker bietet neben seiner Hauptfunktion *Audio Amplifier* (*FBlockID* 22h), ebenfalls die Diagnosefunktion. |
| 0100 | 0101 | 00 | **NetBlock.01.FBlockIDs.SetGet.060001** |
| 0101 | 0100 | 00 | **NetBlock.01.FBlockIDs.Status. 520053000601** |
| 0100 | 0102 | 00 | **NetBlock.02.FBlockIDs.SetGet.060002** |
| 0102 | 0100 | 00 | **NetBlock.02.FBlockIDs.Status. 31000602** |
| 0100 | 0103 | 00 | **NetBlock.03.FBlockIDs.SetGet.060003** |
| 0103 | 0100 | 00 | **NetBlock.03.FBlockIDs.Status. 2200060326002400**<br>Innerhalb eines MOST Rings muss die Kombination aus *FBlockID* und *InstID* eindeutig sein. Dies ist im vorliegenden Beispiel für den Diagnosefunktionsblock 06h noch nicht gegeben, da alle *Slaves* dieselbe *InstID*=00h für diesen Block gemeldet haben. Der *Network Master* löst die Mehrdeutigkeit auf und weist den Slaves die neuen *InstIDs* 01, 02 und 03 für diesen Funktionsblock zu. |
| 0100 | 03C8 | 00 | **NetworkMaster.00.Configuration.Status.OK**<br>Nachdem der *Network Master* die *Central Registry* mit den gemeldeten Funktionsblöcken erfolgreich aufgebaut hat, wird allen Teilnehmern über eine *Broadcast* Nachricht der Konfigurationsstatus *OK* mitgeteilt. Anschließend ist der Ring funktionsbereit. |

**Tab. 3.13** (Fortsetzung)

| Von | Nach | Msg. Type | Steuerbotschaften |
|---|---|---|---|
| 0100 | 0102 | 00 | **AudioDiskPlayer.01.Allocate.StartResult.01** |
| 0102 | 0400 | 03 | **ResourceAllocate.00010203** |
| 0102 | 0100 | 00 | **AudioDiskPlayer.01.Allocate.Result.010200010203**<br>Die *Head Unit* fordert den CD-Spieler auf, Daten seiner logischen Datenquelle 01h zu übertragen. Quelle 01h entspricht in diesem Beispiel dem Stereo-Audiosignal der CD mit Abtastwerten von je 16 bit, d. h. vier physikalischen Kanälen zu je 8 bit. Die Applikation des CD-Spielers fordert diese Kanäle über die *Network Services* an. Die Systemnachricht *Resource Allocate* informiert alle Busteilnehmer, dass diese Kanäle belegt sind. Der CD-Spieler verbindet den Datenstrom der Quelle mit den reservierten MOST Kanälen und bestätigt die erfolgreiche Ausführung des *Allocate*, wobei als Parameter die Quelle (01), die Verzögerung der synchronen Daten relativ zum *Master* (02) und die Liste der belegten Kanäle (00 01 02 03) in der Bestätigung gesendet werden. |
| 0100 | 0103 | 00 | **AudioAmplifier.00.Connect.StartResult.010000010203** |
| 0103 | 0100 | 00 | **AudioAmplifier.00.Connect.Result.01**<br>Damit die übertragenen Audiodaten vom Audioverstärker verarbeitet werden, wird er vom *Network Master* über die Funktion *Connect* aufgefordert, seine logische Datensenke (*Sink*) 01, d. h. seine digitalen Audioeingänge, mit den synchronen Daten auf den Kanälen 0…3 zu verbinden. |
| 0100 | 0102 | 00 | **AudioDiskPlayer.01.DeckStatus.Set.00**<br>Bisher hat der CD-Spieler nur Leerdaten übertragen. Über die Funktion *DeckStatus* erhält er jetzt den Befehl, die CD tatsächlich abzuspielen (Parameter 00=*Play*, 01=*Stop* usw.). |
| 0100 | 0103 | 00 | **AudioAmplifier.00.Volume.Set.1 F** |
| 0100 | 0103 | 00 | **AudioAmplifier.00.Mute.SetGet.0100** |
| 0103 | 0100 | 00 | **AudioAmplifier.00.Mute.Status. 0100**<br>Hier wird die gewünschte Lautstärke (*Volume*) im Verstärker eingestellt und die Stummschaltung (*Mute*) aufgehoben, damit das Audiosignal tatsächlich über die Lautsprecher zu hören ist. |

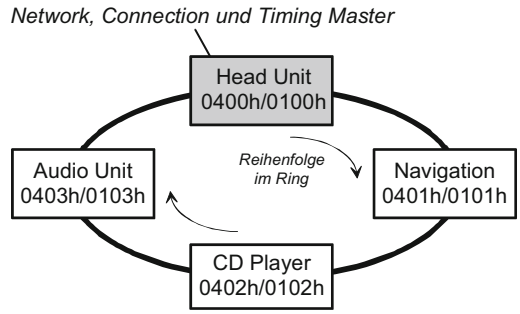


**Abb. 3.39** Beispiel eines MOST25-Rings (mit physikalischen und logischen Adressen)

### 3.4.8 Zusammenfassung MOST

- Bitstrom-orientiertes Übertragungsprotokoll mit Kunststoff-Lichtwellenleiter für Infotainment-Anwendungen, seltener mit elektrischen Zwei-Draht- bzw. Koaxialkabel-Verbindungen, überwiegend in Ringstruktur.
- Bitrate ca. 25 Mbit/s (MOST25), zweite Generation mit 50 Mbit/s (MOST50), Weiterentwicklung mit 150 Mbit/s (MOST150).
- Kommunikation mit logischen Kanälen und reservierter Übertragungsbandbreite (bis zu 60 Byte je MOST25-Botschaft, 117 Byte je MOST50-Botschaft bzw. 372 Byte je MOST150-Botschaft bei einer Botschaftsrate von 48 kHz) mit TDMA-Buszugriff. Nutzdatenrate bei MOST25 bis 2,6 MB/s (MOST25), 5,6 MB/s (MOST50) und 17,8 MB/s (MOST125), wenn keine asynchronen Daten übertragen werden.
- Falls nicht die volle Bandbreite für synchrone Daten benötigt wird, steht der Rest der Bandbreite für eine paketorientierte, asynchrone Übertragung zur Verfügung. Für die paketorientierte Übertragung keine garantierte Botschaftslatenz. Absicherung durch CRC-Prüfsumme. Bei MOST150 können auch Ethernet-Botschaften ohne Umformatierung über MOST übertragen werden.
- Zusätzlich Steuerbotschaften für die Netzwerkverwaltung und Anwendungen mit automatischer Fehlerbehandlung, aber auch ohne garantierte Botschaftslatenz. Datenraten um 50 KB/s [17].

## 3.5 Automotive Ethernet

In den letzten Jahren musste die Automobilindustrie lernen, dass eine Vielzahl inkompatibler und nahezu ausschließlich in der Automobilindustrie verwendeter Lösungen zu hohen Kosten und einem ständigen Weiterentwicklungsaufwand führt. Daher überlegt man im Hinblick auf den weiter wachsenden Bandbreitenbedarf, das im Bürobereich seit vielen Jahren etablierte Kommunikationskonzept Ethernet/IP einzuführen. Ethernet als *Bussystem* (ISO Layer 1 und 2) erscheint aufgrund seiner extrem hohen Stückzahlen kostengünstig und wird wegen seiner hohen Verbreitung auch ständig weiterentwickelt. Die Internetprotokolle IP, TCP, UDP usw. (ISO Layer 3 und 4) erlauben eine transparente Kommunikation, da sie heute von praktisch jedem computerähnlichen Gerät unterstützt werden. Damit vereinfacht sich die Integration von Consumergeräten wie Smartphones erheblich.

### 3.5.1 Ethernet nach IEEE 802.3

Ursprünglich war Ethernet ein Linienbussystem mit CSMA/CD Zugriffsverfahren (Kap. 2). In solchen Systemen steigt die Übertragungslatenz bei höheren Busbelastungen sehr stark an. Moderne Ethernet Netze (Abb. 3.40) werden deshalb in Stern-Topologie ausgelegt, wobei am Kopplungspunkt ein sogenannter *Switch* sitzt, der ankommende Botschaften an

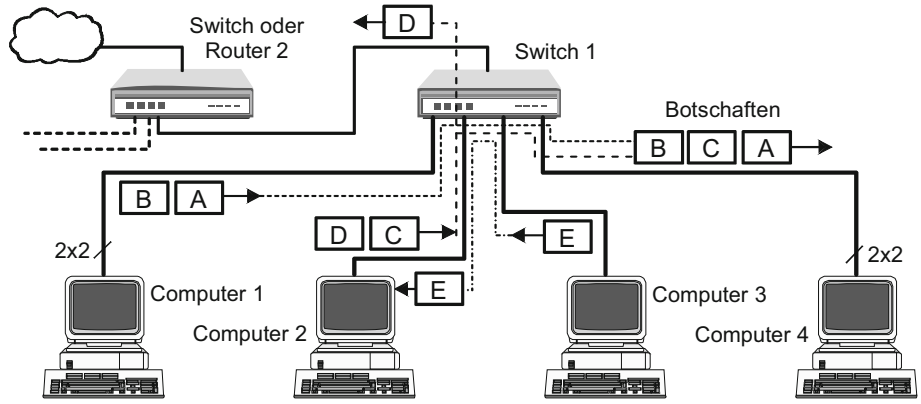


**Abb. 3.40** Topologie eines *Switched Ethernet* Netzes

genau eines der angeschlossenen Geräte weitersendet (*Switched Network*). Der Empfänger wird aus der in den Botschaften enthaltenen Ethernet-Adressen ermittelt. Lediglich bei unbekanntem Empfänger leitet der *Switch* eine Botschaft an alle angeschlossenen Geräte weiter. Welche Geräte angeschlossen sind, ermittelt der *Switch* im laufenden Betrieb aus den Ethernet-Adressen der ankommenden Botschaften selbstständig.

Jeder Computer ist an den *Switch* über ein Paar von Zwei-Draht-Leitungen als Punkt-zu-Punkt Verbindung angeschlossen. Damit ist eine Voll-Duplex-Kommunikation möglich. Die Kopplung zwischen den Leitungen im *Switch* erfolgt nach dem Prinzip des Kreuzschienenverteilers, so dass der *Switch* gleichzeitig Botschaften von verschiedenen Sendern zu mehreren Empfängern übertragen kann (in Abb. 3.40 z. B. Botschaft *A* und *E*). Das System arbeitet kollisionsfrei. Zu größeren Übertragungsverzögerungen kommt es nur, wenn am *Switch* gleichzeitig mehrere Botschaften für denselben Empfänger ankommen. In diesem Fall leitet der *Switch* die erste Botschaft direkt weiter, die restlichen Botschaften werden zwischengespeichert (in Abb. 3.40 z. B. Botschaft *C*, die gleichzeitig mit Botschaft *A* ankommt) und erst mit Verzögerung weitergesendet (*Store and Forward*). Botschaften gehen nur dann verloren, wenn der Pufferspeicher im *Switch* unterdimensioniert ist. Ein Netz kann mehrere *Switche* enthalten. Nach Möglichkeit sollte dabei eine hierarchische Baumstruktur verwendet werden, bei der es zwischen zwei angeschlossenen Geräten jeweils einen eindeutigen Verbindungsweg gibt. Ist dies nicht der Fall, können sich moderne *Switche* mit Hilfe des sogenannten *Spanning Tree Protocols* automatisch so konfigurieren, dass zumindest die logische Verbindungsstruktur eindeutig ist.

Zu Beginn einer Ethernet-Botschaft (Abb. 3.41) wird eine feste Bitfolge, die *Präambel* und der *Start Frame Delimiter SFD*, gesendet, die bei manchen *Physical Layern* zur Taktsynchronisation notwendig sind. Ethernet arbeitet mit Geräteadressierung. Die jeweils 6 Byte langen Ziel- und Quelladressen (*Medium Access Control MAC* Adressen) sind den Ethernet-Kommunikationscontrollern fest zugeordnet und werden von den Chiphersteller weltweit eindeutig festgelegt. Neben den eindeutigen Adressen existieren *Multicast* Adres-

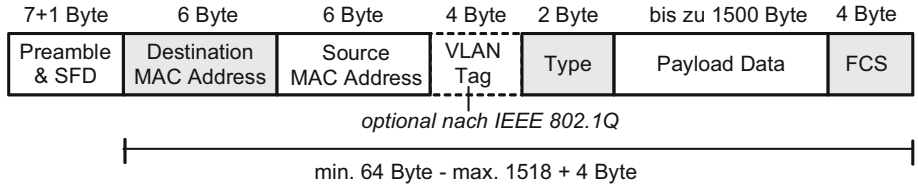


**Abb. 3.41** Format einer Ethernet-Botschaft nach IEEE 802.3

sen für Gerätegruppen sowie eine *Broadcast* Adresse, um eine Botschaft an alle angeschlossenen Geräte gleichzeitig zu senden. Das optionale VLAN Tag erlaubt die Bildung von Unternetzen, den virtuellen LANs. Die Teilnehmer eines VLAN können miteinander kommunizieren, sind aber für andere Teilnehmer außerhalb des Unternetzes praktisch unsichtbar, weil ihre Botschaften nur innerhalb des eigenen Unternetzes weitergeleitet werden. Zusätzlich zur VLAN-Kennung kann das Tag-Feld auch eine Prioritätsforderung enthalten, die von einem *Switch* berücksichtigt wird, wenn mehrere ankommende Botschaften zur Weiterleitung anstehen.

Durch ein Typfeld wird der Inhalt des folgenden Nutzdatenfeldes gekennzeichnet. Enthält dieses eine IPv4-Botschaft (Internet Protokoll Version 4), so wird z. B. $800_H$ gesendet, bei IPv6 $86DD_H$. Das Datenfeld kann maximal 1500 Byte enthalten. Sollen nur wenige Daten versendet werden, muss das Nutzdatenfeld gegebenenfalls mit Leerdaten aufgefüllt werden, weil die Ethernet-Botschaft (ohne Präambel und SFD) mindestens 64 Byte lang sein muss. Am Ende der Botschaft folgt eine Prüfsumme, die *Frame Check Sequence*. Bei Übertragungsfehlern verwirft der Empfänger die Botschaft automatisch, eine Fehlermeldung oder Empfangsbestätigung erfolgt nicht. Zwischen zwei Ethernet-Botschaften muss ein Mindestabstand (*Inter Packet Gap oder Inter Frame Spacing*) von 96 Bittakten eingehalten werden.

Zum Erfolg von Ethernet hat die ausgeprägte Trennung zwischen *Data Link Layer* (im Ethernet-Jargon *Medium Access Control MAC*) und der Bitübertragungsschicht (*Physical Layer PHY*) beigetragen (Abb. 3.42). Beide sind in IEEE 802.3 standardisiert, wobei die MAC Schicht seit vielen Jahren praktisch unverändert ist, während die PHY Ebene laufend weiterentwickelt wird, um höhere Übertragungsraten abzudecken. Der Durchbruch gelang Ethernet mit einer Bitrate von 10 Mbit/s, die zunächst über Koaxialkabel und später über ungeschirmte, verdrillte Leitungspaare übertragen wurde (Ethernet-Bezeichnung 10-Base-T). Die nächste Ausbaustufe verwendet 100 Mbit/s (*Fast Ethernet* 100-Base-TX) mit zwei verdrillten Leitungspaaren, während heute im Bürobereich oft schon 1 Gbit/s (*Gigabit Ethernet* 1000-Base-T) mit vier verdrillten Leitungspaaren eingesetzt wird. Für noch höhere Bitraten (10 bzw. 100 Gbit/s und mehr) gibt es ebenfalls schon erste Komponenten. Daneben existieren verschiedene Untervarianten, aber auch Versionen für Glasfaserkabel, Varianten für Kleingeräten mit integrierter Spannungsversorgung über die Ethernet-Anbindung (*Power over Ethernet*) oder umgekehrt Ethernet-Verbindungen über das gewöhnliche 230 V Hausnetz (*Powerline Ethernet*).

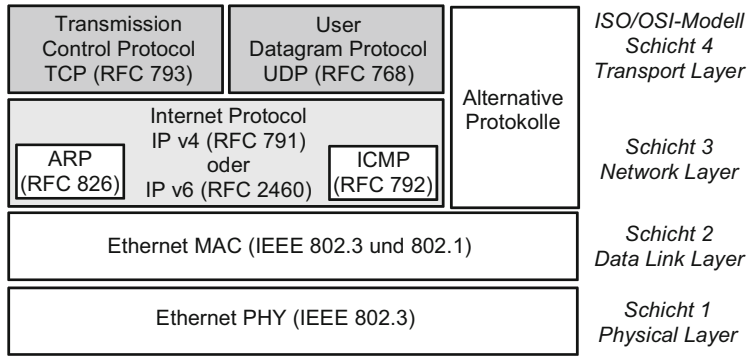


**Abb. 3.42** Protokollstapel Ethernet, IP, TCP und UDP

Die Trennung zwischen MAC und PHY Schicht ist auch in der Hardwareimplementierung üblich (Abb. 3.43). Die PHY-Komponente besteht bei Ethernet nicht nur aus einem einfachen *Transceiver*, sondern enthält die komplette Logik zur Bitübertragung. Da die Leitungsschnittstelle galvanisch entkoppelt wird, sind zusätzlich zwei kleine Übertrager und einige passive Bauteile notwendig. Die meisten PHY-Komponenten können selbstständig erkennen, ob die Kommunikationspartner mit 10 Mbit/s oder mit 100 Mbit/s arbeiten (*Autonegotiation*). Die MAC Schicht ist oft bereits in den Mikrocontroller integriert. Die Signale zwischen MAC und PHY-Schicht sind als *Media Independent Interface MII* weitgehend standardisiert, so dass Komponenten von unterschiedlichen Herstellern kombiniert werden können.

### 3.5.2 Autotauglicher Physical Layer BroadR-Reach

Wegen des Kabels mit zwei Leitungspaaren und der Steckverbindertechnik gelten Standard-Ethernet-Komponenten weder bezüglich der Kosten noch im Hinblick auf die EMV-, Temperatur-, Schüttel- und Feuchtebelastung als direkt autotauglich. Bei der neuen Diagnoseschnittstelle *Diagnostic over Ethernet DoIP* (siehe Abschn. 4.6) wird die konventionelle Technik trotzdem eingesetzt, weil diese Anforderungen im Werkstatt- und Fertigungsumfeld unproblematisch sind. Für den Einsatz in der On-Board-Kommunikation

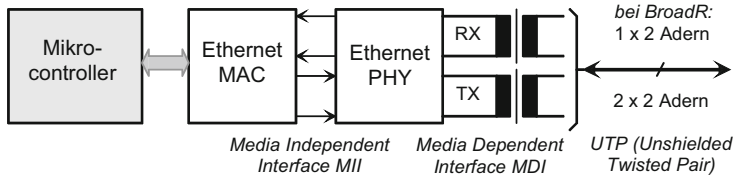


**Abb. 3.43** Typischer Aufbau einer 10/100-Base-T Ethernet-Schnittstelle (*vereinfacht*)

dagegen soll ein spezieller *Physical Layer* der Firma Broadcom verwendet werden. Deren unter der Bezeichnung *BroadR-Reach* vermarktete PHY-Bausteine erlauben derzeit eine Bitrate von 100 Mbit/s im Voll-Duplex-Betrieb über eine einfache, verdrillte Zwei-Draht-Leitung. Die Anforderungen an das Kabel und die Steckverbinder sollen dabei nicht höher sein als bei einem FlexRay-System. Unter dem Dach der *One-Pair Ethernet (OPEN) Alliance* haben sich Broadcom und verschiedene Automobilhersteller und Zulieferer zu einem Konsortium zusammengeschlossen, das das Konzept vollständig standardisieren und in der Praxis erproben will. Mindestens einer der beteiligten Automobilhersteller hat bereits Serienprojekte auf dieser Basis angekündigt, wobei zunächst MOST und später auch FlexRay durch Ethernet ergänzt bzw. ersetzt werden soll.

Wie erfolgreich *BroadR-Reach* sein wird, hängt sicher nicht nur davon ab, ob die bisher proprietäre Technologie auch von anderen Halbleiterherstellern nachgebaut werden kann und darf. Bei MOST war ja genau dies ein erhebliches Hindernis. Entscheidend für Ethernet aber wird sein, ob es gelingt, ein kostengünstiges Konzept für die Echtzeitfähigkeit zu erarbeiten.

### 3.5.3 Echtzeitfähigkeit mit IEEE 802.1 Audio-Video-Bridging AVB

Die Dauer einer Ethernet-Botschaft lässt sich abschätzen zu

$$T_{\text{Frame}} \approx \left(n_{\text{Header+Trailer}} + n_{\text{Data}}\right) \cdot T_{\text{bit}}. \tag{3.25}$$

Der *Overhead* der Botschaft ist mit $n_{\text{Header+Trailer}} = 30\,\text{Byte} = 240\,\text{bit}$ sehr groß, wenn man die Präambel und das optionale VLAN-Tag mit berücksichtigt. Die kürzestmögliche Ethernet-Botschaft (Präambel plus 64 Byte, davon 42 Byte Daten) wird bei einer Bitrate von $f_{\text{bit}} = 1 / T_{\text{bit}} = 100\,\text{Mbit/s}$ in knapp 6 µs übertragen. Bei $n_{\text{Data}} = 254\,\text{Byte}$ Daten, der Maximallänge von FlexRay, benötigt die Botschaft 23 µs und selbst die längste Ethernet-Botschaft mit $n_{\text{Data}} = 1500\,\text{Byte}$ lässt sich in nur 123 µs versenden.

Die maximale Bandbreite einer Ethernet-Verbindung ergibt sich zu

$$f_{\text{Data}} = \frac{n_{\text{Data}}}{T_{\text{Frame}} + 96 \cdot T_{\text{bit}}}, \tag{3.26}$$

wobei berücksichtigt wurde, dass zwischen den Botschaften eine Pause von mindestens 96 bit eingehalten werden muss. Ein 100 Mbit/s Netz hat damit bei einer Datenlänge zwischen 42 und 1500 Byte eine Bandbreite von 6 bis 12 MB/s.

Im Gegensatz zur Dauer einer Botschaft lässt sich die gesamte Übertragungsverzögerung $T_{\text{Latenz}}$ zwischen Sender und Empfänger nicht so einfach ermitteln. Zur Dauer der Ethernet-Botschaft selbst addiert sich die Durchlaufverzögerung aller *Switche* auf dem Weg zwischen Sende- und Empfangssteuergerät.

$$T_{\text{Latenz}} = T_{\text{Frame}} + \sum T_{\text{Switch}} \tag{3.27}$$

Die Durchlaufverzögerung $T_{\text{Switch}}$ hängt zunächst von deren interner Arbeitsweise ab. Jeder *Switch* muss die Botschaft mindestens solange verzögern, bis er die Zieladresse empfangen hat und erkennen kann, an welches Steuergerät er die Botschaft weiterleiten muss (*Cut Through*). Die theoretisch minimale Durchlaufzeit lässt sich daher zu $T_{\text{Switch,min}} = 112 \cdot T_{\text{bit}} = 1{,}2\,\mu\text{s}$ abschätzen. Dieser Wert wird in der Praxis aber nicht erreichbar sein, da die Überprüfung der Adresse und das möglich Umkopieren der Botschaft zwischen verschiedenen Pufferspeichern zusätzliche Zeit kostet. Hersteller industrieller *Cut Through Switche* geben Werte im Bereich von 5 µs an. Manche *Switche* warten aber sogar mit der Weiterleitung, bis sie eine Botschaft vollständig empfangen haben (*Store and Forward*). In diesem Fall kann nämlich die Prüfsumme der empfangenen Botschaft überwacht und fehlerhafte Botschaften blockiert werden. Solche *Switche* haben eine Verzögerung von mindestens $T_{\text{Switch}} = T_{\text{Frame}}$.

Kommen an einem *Switch* gleichzeitig auch noch mehrere Botschaften für denselben Empfänger an, so werden diese Botschaften in jedem Fall vollständig zwischengespeichert und erst nacheinander weitergeleitet. Sowohl die zulässige Durchlaufverzögerung als auch die Reihenfolge bei der Weiterleitung sind im Ethernet-Standard IEEE 802.3 nicht geregelt, so dass Ethernet in seiner Grundform nicht echtzeitfähig ist.

Um trotzdem Anwendungen mit Echtzeit-ähnlichen Anforderungen wie beispielsweise *Voice-over-IP-Telefonie* oder *Live-Video-Streams* im Internet zu realisieren, behilft man sich mit speziellen Protokollen in den höheren Ebenen des Protokollstapels, z. B. dem *Real Time Transport Protocol RTP* nach RFC 3550 . Diese Lösungen können aber nur eine ausreichende mittlere Bandbreite für die Datenübertragung sicherstellen. Das Problem der variablen Übertragungsverzögerung (*Jitter*) selbst können sie nicht beseitigen, sondern nur kaschieren. Der entscheidende *Trick* besteht darin, im Empfänger einen ausreichend großen Puffer einzurichten, in dem die mit variabler Verzögerung ankommenden Botschaften zwischengespeichert und mit konstanter Rate ausgelesen und weiterverarbeitet werden. Dadurch wird die Gesamtverzögerung zwischen Sender und Empfänger zwar größer, erscheint aber als konstante Latenz. Für Sprach- und Bildübertragungen ist dies unkritisch, zumal dort sogar sporadisch ausfallende Botschaften vom Menschen nicht wahrgenommen werden.

Bei zeitkritischen Steuerungsdaten oder gar in geschlossenen Regelkreisen muss die Übertragung aller Botschaften mit bekannter Maximalverzögerung gewährleistet werden. Nicht übertragene Botschaften können dort sogar sicherheitskritisch werden. Für Industriesteuerungen, bei denen Ethernet schon seit längerem eingesetzt wird, wurden daher eine Reihe von Echtzeitlösungen wie *ProfiNet*, *EtherCAT*, *EtherNet/IP*, *Modbus/TCP, Sercos III* oder *TTEthernet* entwickelt [19]. Hinter vielen dieser Lösungen steht allerdings nur ein einzelner großer Automatisierungshersteller oder ein Konsortium von kleineren Firmen. Ein einheitlicher Industriestandard fehlt. Gelegentlich werden sogar modifizierte *Data Link Layer* und damit herstellerspezifische Kommunikationscontroller verwendet. Um eine kostengünstige, für den Massenmarkt taugliche Lösung zu finden, haben sich *Consumer*-Geräte- und Computerhersteller zusammengeschlossen und ein Konzept für das Audio-Video-Bridging AVB erarbeitet, die Echtzeit-Übertragung von Audio- und Videodaten direkt über den Ethernet *Data Link Layer*. Zur Unterstützung des Konzepts, das

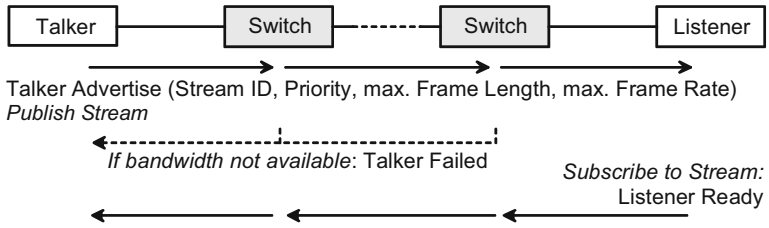


**Abb. 3.44** Einrichten einer Echtzeitverbindung mit dem *Stream Reservation Protocol*

inzwischen unter IEEE 802.1 standardisiert ist, haben sich diese Hersteller in der *AVnu Alliance* organisiert. Diesem Konsortium haben sich mittlerweile auch die Automobilhersteller und Zulieferer der *OPEN Alliance* Initiative sowie einige Automatisierungshersteller angeschlossen, um Ethernet vollständig echtzeit- und automobiltauglich zu machen [20].

AVB ist als IEEE 802.1Q standardisiert, die Basisnorm enthält seit 2012 auch die früheren Teilnormen 802.1Qat und 802.Qav. IEEE 802.1Q spezifiziert u. a. das *Multiple Stream Reservation Protocol MSRP*, mit dem Übertragungskanäle (*Streams*) mit garantierter Bandbreite und maximaler Übertragungsverzögerung eingerichtet werden können. Die Norm nennt bis zu 75 % der Bandbreite, die für AVB-Botschaften reserviert werden sollen, und strebt eine Gesamtlatenz von unter 2 ms bei bis zu sieben Teilstrecken (*Hops*), also sechs *Switchen* zwischen Sender und Empfänger an.

Ein Gerät, das Echtzeitdaten senden möchte, in der Norm als *Talker* bezeichnet, kündigt seinen Bandbreitenbedarf mit Hilfe einer *Talker Advertise* Botschaft an (Abb. 3.44). Der angeschlossene *Switch* prüft, ob der erforderliche Pufferspeicher (*Queue*) verfügbar ist und leitet die Botschaft an alle anderen Geräte weiter, sofern er die Bandbreitenforderung erfüllen kann. Dabei ergänzt er die Botschaft um einen Schätzwert für seine eigene Durchlaufverzögerung, so dass der Empfänger eine Information über die gesamte Übertragungslatenz der späteren Echtzeitdaten bekommt. Falls die Bandbreitenforderung nicht erfüllbar ist, teilt der *Switch* dies durch eine *Talker Failed* Botschaft mit. Ein Steuergerät, das Echtzeitdaten empfangen möchte, in der Norm als *Listener* bezeichnet, meldet sich durch eine *Listener Ready* Botschaft beim *Talker* an. Im Fehlerfall wird eine *Listener Failed* Botschaft versendet. Sobald eine *Talker Advertise* Ankündigung durch mindestens eine *Listener Ready* Anmeldung bestätigt wurde, reservieren alle beteiligten Geräte die Bandbreite verbindlich und die Übertragung kann beginnen. Ein *Stream* bleibt aktiv, bis er wieder abgekündigt wird (*Publish-Subscriber* Konzept nach Kap. 2). Die Echtzeitbotschaften ebenso wie die Botschaften des MSRP-Protokolls sowie einiger weiterer Hilfsprotokolle werden als gewöhnliche Ethernet-Botschaften versendet (Abb. 3.41), wobei das *Type* und das *VLAN Tag* Feld verwendet werden, um zwischen den verschiedenen Protokollen, *Streams* und deren Prioritäten zu unterscheiden. Da die Echtzeitbotschaften in einem *Switch* priorisiert werden, können in einem IEEE 802.1Q kompatiblen Netz auch gewöhnliche Botschaften transportiert werden, ohne die Echtzeitkommunikation zu stören.

Das Problem, dass auch eine höherpriore Botschaft warten muss, wenn ein *Switch* gerade auf derselben ausgehenden Verbindung eine andere Botschaft überträgt, kann aber natürlich grundsätzlich nicht gelöst werden. Die einzige mögliche Abhilfe dafür ist, im gesamten Netz ein synchrones Übertragungsschema einzuführen, das solche Wartesituationen durch zeitversetztes Senden bereits an der Quelle vermeidet. FlexRay hat aus diesem Grund den beschriebenen zeitgesteuerten Kommunikationsplan (*Schedule*) mit fest zugewiesenen Zeitschlitzen (*Slots*) und einer netzweiten Zeitbasis (*Cycles*, *Macro Ticks*). Ethernet dagegen verwendet wie CAN ein ereignisgesteuertes Sendekonzept und besitzt keine eigene netzweite Zeitbasis. Mit Hilfe der in IEEE 802.1AS definierten Untermenge des aus IEEE 1588 schon länger bekannten *Precision Time Protocol PTP* kann diese netzweite Zeitbasis nachgerüstet werden.

Für die Zeitsynchronisation wird das Netz mit Hilfe des *Best Master Clock Algorithmus* dynamisch in einer Baumstruktur organisiert, wobei als Wurzel der Teilnehmer mit der genauesten Zeitbasis festgelegt wird. Dieser Teilnehmer wird als *Grandmaster* bezeichnet. Auf dessen Zeitbasis synchronisieren sich nun die Mitglieder der nächsten Hierarchiestufe, die dabei als *Slaves* agieren. Gegenüber der nächsttieferen Hierarchiestufe dreht sich ihre Rolle um und sie werden zum Master. Auf diese Weise wird die Zeitbasis stufenweise von der Wurzel bis zu den Spitzen des Baums synchronisiert, wobei in jedem Teilschritt nur jeweils zwei Teilnehmer beteiligt sind, einer in der *Master*-, der andere in der *Slave*-Rolle. Kern des Verfahrens sind die Botschaften *Sync* und *DelayRequest* (Abb. 3.45), deren Sende- bzw. Empfangszeitpunkte jeweils mit Hilfe der lokalen Zeitbasen gemessen werden. Die vom Master ermittelten Werte werden dem *Slave* in zwei zusätzlichen Botschaften *FollowUp* und *DelayResponse* übermittelt. Aus den vier Messwerten kann der Slave nun sowohl den Zeitversatz seiner eigenen Uhr gegenüber der des Masters als auch die Übertragungszeit der Botschaften berechnen. Dabei wird angenommen, dass die Übertragungsdauer konstant und in beiden Richtungen gleich ist. Da die lokalen Uhren driften können, wird der Vorgang periodisch wiederholt. Die Genauigkeit des Verfahrens hängt entscheidend davon ab, wie genau der Sende- bzw. Empfangszeitpunkt der Ethernet-Botschaften tatsächlich gemessen werden kann. Erfolgt die Messung mit Hilfe von Software-Interrupts, so sind bei den meisten Mikrocontrollern Fehler von deutlich über 10 µs zu erwarten. Für geringfügig modifizierte Ethernet-Controller, die automatisch interne Hardware-Zeitstempel (*Timestamp*) erzeugen, geben Hersteller dagegen erreichbare Fehler bei der Zeitsynchronisation im unteren Mikrosekundenbereich an.

### 3.5.4 Höhere Protokollschichten IP, TCP und UDP

Theoretisch könnten Anwendungen ihre Daten direkt im Datenfeld der Ethernet-Botschaften verpacken. Da Ethernet aber keine Segmentierung und keine Fehlerbehandlung unterstützt und für Weitverkehrsnetze und Funkübertragungen nicht direkt einsetzbar ist, verwenden Anwendungen heute einige höhere Protokollebenen.

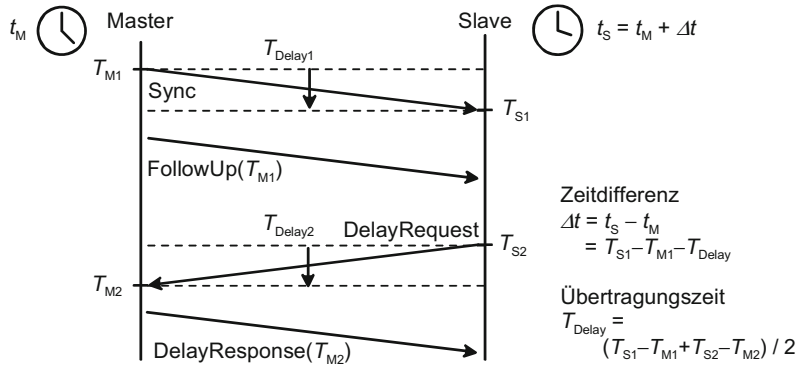


**Abb. 3.45** Prinzip der Zeitsynchronisation nach IEEE 802.1AS

Auf Schicht 3 des ISO/OSI-Modells (Abb. 3.42) verwendet man das *Internet Protocol IP* nach RFC 791/2460. Dieses benutzt gerätebezogene Adressen, die fest oder mittels des Hilfsprotokolls *Dynamic Host Configuration Protocol DHCP* nach RFC 2131 auch dynamisch vergeben werden können. Damit Rechner weltweit miteinander kommunizieren können, sollten die Adressen eindeutig sein und werden daher von der Nutzerorganisation ICANN zentral an *Internet Service Provider ISP* und von diesen an die Endteilnehmer vergeben. Leider waren für diese Adressen ursprünglich nur 32 bit Werte vorgesehen, die anfangs zudem sehr großzügig verteilt wurden, so dass der Adressbereich mittlerweile ausgeschöpft ist. Aus diesem Grund wird die lange Zeit stabile Protokollversion IPv4 allmählich durch die neuere Version IPv6 abgelöst, die 128 bit IP Adressen vorsieht. Leider vergrößert sich die Länge des Botschaftsheaders dabei von 20 Byte auf 40 Byte. Die Zuordnung der IP Adressen zu den Ethernet MAC Adressen kann innerhalb eines Ethernet-Netzes mit den Hilfsprotokollen *Address Resolution ARP* bzw. *Neighbor Discovery Protocols NDP* (RFC 826/4861 NDP) ermittelt werden. Die numerischen IP-Adressen, z. B. 134.108.34.3, sind für Menschen recht unhandlich. Daher hat man Klarnamen wie www.hs-esslingen.de eingeführt, deren Zuordnung zu den IP-Adressen von Web-Browsern und ähnlichen Anwendungen im *Domain Name System DNS*, einer Art Telefonbuch für Internet-Adressen, abgefragt werden kann. Der Datenbereich einer IP-Botschaft kann bis zu 64 KB lang sein. Theoretisch können IP-Botschaften zwar bei der Weiterleitung segmentiert werden, in der

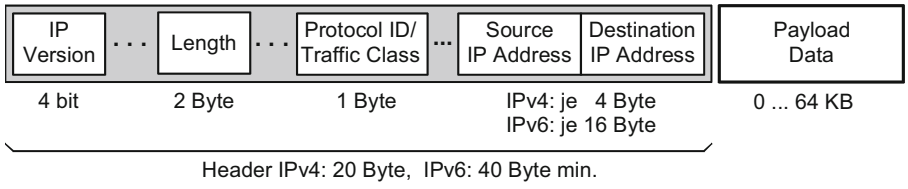


**Abb. 3.46** Aufbau einer IP Botschaft (*vereinfacht*)

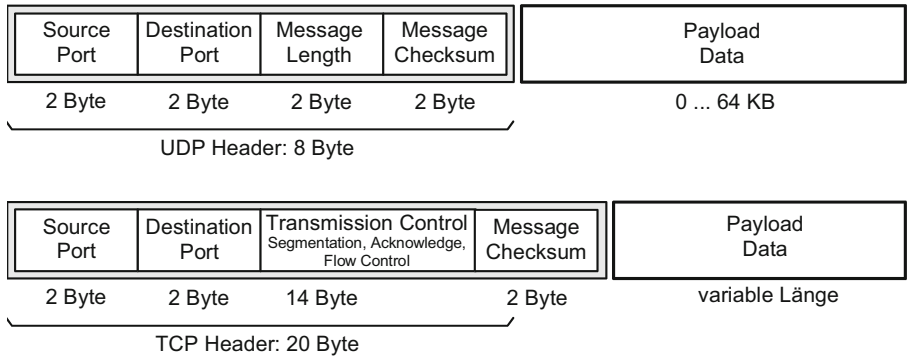


**Abb. 3.47** UDP (*oben*) und TCP Botschaften (*vereinfacht*)

Praxis wird man die Länge aber meist so beschränken, dass die IP-Botschaft vollständig in das Datenfeld einer Ethernet-Botschaft passt. Falls eine IP-Botschaft nicht weitergeleitet werden kann, z. B. weil eine falsche IP-Adresse verwendet wurde, erhält der Sender von der Station, die den Fehler feststellt, eine Fehlermeldung mittels des Hilfsprotokolls *Internet Control Message Protocol*. Die weitere Fehlerbehandlung ist jedoch auf der IP Ebene genauso wenig definiert wie die Überwachung des Datenfelds durch eine Prüfsumme. IP erlaubt also lediglich eine verbindungslose ungesicherte Übertragung. Das *Protocol ID* Feld zeigt an, welches Protokoll die IP-Botschaft transportiert, z. B. 1 für ICMP, 6 für TCP oder 17 für UDP.

In der nächsthöheren Schicht 4 sind mit UDP und TCP zwei alternative Protokolle im Einsatz (Abb. 3.42). Zusätzlich zu den gerätebezogenen IP-Adressen der Schicht 3 werden in der Schicht 4 sogenannte *Ports* verwendet, die eine anwendungsbezogene Adressierung erlauben. Die Portnummern sind auf der Serverseite meist fest vergeben, z. B. $80_H$ für Web-Server (*Well-Known Port*), auf der Clientseite werden sie in der Regel dynamisch festgelegt. Durch die sogenannten *Sockets*, die Kombination der IP-Adresse und der Portnummer, kann eine Anwendung auf einem Rechner gezielt Verbindung zu einer Anwendung auf einem anderen Rechner aufnehmen.

Das *User Datagram Protocol UDP* nach RFC 768 erlaubt eine verbindungslose, unbestätigte Datenübertragung von Datenblöcken bis zu 64 KB, die über eine Prüfsumme gesichert werden (Abb. 3.47). Damit ist es möglich, fehlerhafte Botschaften auf der Empfangsseite zu erkennen und zu ignorieren, eine Rückmeldung an den Sender und eine automatische Wiederholung gibt es aber nicht.

Das *Transmission Control Protocol TCP* nach RFC 793 (Abb. 3.47) dagegen kann beliebig große Datenblöcke segmentiert übertragen, deren Empfang bestätigt und im Fehlerfall wiederholt wird [21]. Die Verbindung muss vor Beginn der Übertragung geöffnet und am Ende wieder abgebaut werden. Die Segmentgröße wird eventuell dynamisch angepasst, gegebenenfalls werden sogar mehrere Datenblöcke zwischengespeichert, bevor eine Botschaft versendet wird, um den relativen Overhead zu verringern. Alle diese Vorgänge erfolgen

**Tab. 3.14** Bandbreitenschätzung für Ethernet/IP (Ethernet mit 100 Mbit/s)

| | Overhead in Byte je Botschaft | Max. Nutzdatenrate in MB/s | | |
|---|---|---|---|---|
| | | Nutzdaten je Botschaft | | |
| | | 64 Byte | 254 Byte | 1024 Byte |
| Ethernet 100 Mbit/s | 30 | 7,5 | 10,7 | 12,0 |
| IPv4 auf Ethernet | 30 + 20 = 50 | 6,3 | 10,0 | 11,8 |
| IPv6 auf Ethernet | 30 + 40 = 70 | 5,5 | 9,4 | 11,6 |
| UDP/IPv6 auf Ethernet | 30 + 40 + 8 = 78 | 5,2 | 9,2 | 11,5 |
| TCP/IPv6 auf Ethernet | 30 + 40 + 20 = 90 | 4,8 [a] | 8,9 [a] | 11,4 [a] |

[a] Echtzeittauglichkeit siehe Text.

für die Anwendung weitgehend unsichtbar und sind von dieser nur wenig beeinflussbar. Das Zeitverhalten des TCP Protokolls ist daher nicht streng deterministisch und dürfte für Echtzeitanwendungen nur bedingt geeignet sein.

Ethernet bzw. UDP/IP auf Ethernet dagegen sind prinzipiell einsetzbar, wobei allerdings klar sein muss, dass die in Tab. 3.14 angegebenen Nutzdatenraten für Steuergeräte eher theoretischen Wert haben dürften. Noch deutlich mehr als schon bei FlexRay wird in der Praxis nicht das physikalische Übertragungssystem sondern der Software-Protokollstack in den Steuergeräten die tatsächlichen Latenzzeiten und den erreichbaren Durchsatz bestimmen. Während auf der Ethernet-Ebene wie bei CAN und FlexRay die Botschaftsheader und Prüfsummen weitgehend automatisch durch die Hardware im Kommunikationscontroller erzeugt werden, müssen die IP, UDP und TCP Header sowie die eigentlichen Nutzdaten durch die Software zeitrichtig bereitgestellt und weiterverarbeitet werden. Ohne große RAM-Speicher zum Zwischenspeichern der Botschaften und schnelle Mikroprozessoren zur Berechnung und Überprüfung der Prüfsummen kann die mögliche Bandbreite nicht genutzt werden. Während solche Rechenleistungen bei High-End-Infotainmentsystemen auch im Fahrzeug schon verfügbar sind, müssen Steuergeräte im Antriebs- und Fahrwerksbereich noch deutlich aufgerüstet oder weite Teile des Protokollstacks in Hardware implementiert werden. Und in letzter Konsequenz sollten ähnlich wie bei Nutzfahrzeugen mit SAE J1939/71 (siehe Abschn. 4.5) endlich auch im PKW-Bereich die übertragenen Echtzeitdaten und nicht nur das Übertragungssystem standardisiert werden.

## 3.6 Normen und Standards zu Kapitel 3

| | |
|---|---|
| CAN | Bosch: CAN Specification Version 2.0, 1991, www.can.bosch.com<br>ISO 11898-1 Road Vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signalling, 2003 und 2006, www.iso.org<br>ISO 11898-2 Road Vehicles – Controller area network (CAN) – Part 2: High-speed medium access unit, 2003, www.iso.org<br>ISO 11898-3 Road Vehicles – Controller area network (CAN) – Part 3: Low-speed fault tolerant medium dependent interface, 2006, www.iso.org<br>ISO 11898-4 Road Vehicles – Controller area network (CAN) – Part 4: Time triggered communication, 2004, www.iso.org<br>ISO 11898-5 Road Vehicles – Controller area network (CAN) – Part 5: High-speed medium access unit with low-power mode, 2007, www.iso.org<br>ISO 11898-6 Road Vehicles – Controller area network (CAN) – Part 6: High-speed medium access unit with selective wakeup-functionality, 2013, www.iso.org<br>ISO 11992-1 bis -4 Road vehicles – Interchange of digital information on electrical connections between towing and towed vehicles, 2003 bis 2008<br>SAE J2284/1 High Speed CAN for Vehicle Applications at 125 Kbps, 2002, www.sae.org<br>SAE J2284/2 High Speed CAN for Vehicle Applications at 250 Kbps, 2002, www.sae.org<br>SAE J2284/3 High Speed CAN for Vehicle Applications at 500 Kbps, 2010, www.sae.org<br>SAE J2411 Single Wire CAN Network for Vehicle Applications, 2000, www.sae.org<br>ISO 15765 siehe Kap. 5<br>SAE J1939 siehe Kap. 4<br>Bosch: CAN with Flexible Data-Rate. Specification V1.0, 2012, www.can.bosch.com |
| LIN | LIN Specification Package Revision 1.3. 2002, www.lin-subbus.org<br>LIN Specification Package Revision 2.0. 2003, www.lin-subbus.org<br>LIN Specification Package Revision 2.1, 2006, www.lin-subbus.org<br>LIN Specification Package Revision 2.2 A, 2010, www.lin-subbus.org<br>ISO 17987-1 bis -7 Road vehicles – Local Interconnect Network (LIN), in Vorbereitung, www.iso.org<br>SAE J2602/1 LIN Network for Vehicle Applications, 2012, www.sae.org<br>SAE J2602/2 Conformance Testing for SAE J2602 LIN, 2012, www.sae.org<br>SAE J2602/3 File Structures for a Node Capability File (NCF), 2010, www.sae.org |
| MOST | MOST Specification Framework, Rev. 1.1, 1999, www.mostcooperation.com<br>MOST Specification Rev. 2.5, 2006, www.mostcooperation.com<br>MOST Specification Rev. 3.0, 2008, www.mostcooperation.com<br>MOST Specification Rev. 3.0 E2, 2010, www.mostcooperation.com<br>MOST Dynamic Specification Rev. 3.0, 2013, www.mostcooperation.com<br>MOST MAMAC Specification Rev. 1.1, 2005, www.mostcooperation.com<br>MOST Core Compliance Test Specification Rev. 1.2, 2006, www.mostcooperation.com<br>MOST Function Block Library für Netzknoten mit Slave- oder Masterfunktion, Audio Verstärker, Disk Player usw. Jeweils einzeln versioniert. www.mostcooperation.com |

| | |
|---|---|
| FlexRay | *FlexRay Consortium* ( www.flexray.com)<br>FlexRay Communications System – Protocol Specification: Version 2.1 Revision A, 2005; Version 3.0.1, 2010<br>FlexRay Communications System – Electrical Physical Layer Specification: Version 2.1 Revision B, 2006; Version 3.0.1, 2010<br>FlexRay Communications System – Electrical Physical Layer Application Notes: Version 2.1 Revision B, 2006; Version 3.0.1, 2010<br>FlexRay Communications System – Preliminary Node-Local Bus Guardian Specification Version 2.0.9, 2005, www.flexray.com<br>FlexRay Communications System – Preliminary Central Bus Guardian Specification Version 2.0.9, 2005, www.flexray.com<br>*ISO (* www.iso.org )<br>ISO 17458 Road vehicles - FlexRay communications system<br>Part 1: General information and use case definition, 2013<br>Part 2: Data link layer Specification, 2013<br>Part 3: Data link layer conformance test specification, 2013<br>Part 4: Electrical physical layer specification, 2013<br>Part 5: Electrical physical layer conformance test specification, 2013<br>ISO 10681 Road vehicles – Communication on FlexRay, siehe Kap. 5<br>SAE J2813 FlexRay for Vehicle Applications, noch nicht veröffentlicht, www.sae.org |
| Ethernet | IEEE 802.3 Local and metropolitan area networks. Part 3: Carrier Sense Multiple Access with Collision Detection Access Method (CSMA/CD) and Physical Layer Specifications, 2008, www.ieee.org<br>IEEE 802.1Q Local and metropolitan area networks – Media Access Control Bridges and virtual Brdged Local Area Networks, 2011, www.ieee.org<br>IEEE 802.1AS Local and metropolitan area networks – Timing and Synchronisation for Time-Sensitive Applications in Bridged Local Area Networks, 2011<br>IEEE 802.1BA Local and metropolitan area networks – Audio Video Bridging Systems, 2011, www.ieee.org<br>IEEE 1588 Precision Clock Synchronisation Protocol for Networked Measurement and Control Systems, 2008, www.ieee.org<br>IEEE 1722 Layer 2 Transport Protocol for Time-Sensitive Applications in a Bridged Local Area Network, 2011, www.ieee.org<br>IEEE 1733 Layer 3 Transport Protocol for Time-Sensitive Applications in Local Area Networks, 2011, www.ieee.org |
| TCP/IP<br>UDP | Internet Protokolle IP, TCP, UDP u. a. siehe Kap. 8<br>Internet Engineering Task Force IETF, www.ietf.org |

## Literatur

[1] K. Etschberger: Controller Area Network. Hanser Verlag, 3. Auflage, 2006

[2] W. Lawrenz, N. Obermöller: CAN Controller Area Network. VDE Verlag, 5. Auflage, 2011

[3] Hartwich, F.: CAN with Flexible Data-Rate. 13. International CAN Conference 2012. www.can.bosch.com

[4] K. Tindell, A. Burns, A. Wellings: Calculating CAN Message Response Times. Control Engineering Practice, Heft 8, 1995, S. 1163-1169

[5] A. Burns, R. Davis, R. Bril, J. Lukkien: CAN Schedulability Analysis: Refuted, Revised and Revisited. Real-Time Systems Journal, Springer Verlag, Heft 3, 2007, S. 239-272

[6] T. Nolte, H. Hansson, C. Norström, S. Punnekkat: Using Bit-stuffing Distributions in CAN Analysis. IEEE Real-time Embedded Systems Workshop, London, Dez. 2001

[7] S. Punnekkat, H. Hansson, C. Norström: Response Time Analysis under Errors for CAN. IEEE Real-Time Technology and Applications Symposium, Juni 2000, S. 258-265

[8] N. Navet, F. Simonot-Lion: Automotive Embedded Systems Handbook. CRC Press, 1. Auflage, 2009

[9] K. Tindell: Adding Time-Offsets to Schedulability Analysis. Technical Report YCS 221, University of York, 1994

[10] A. Grzemba, H.-C. von der Wense: LIN-Bus. Franzis Verlag, 1. Auflage, 2005

[11] J. Berwanger, M. Peteratzinger, A. Schedl: FlexRay-Bordnetz für Fahrdynamik und Fahrerassistenzsysteme. Sonderausgabe electronic automotive, 2008

[12] M. Rausch: FlexRay. Grundlagen, Funktionsweise, Anwendung. Hanser Verlag, 1. Auflage, 2007

[13] D. Paret: FlexRay and its Applications. John Wiley & Sons Verlag, 2. Auflage, 2012

[14] N.N.: E-Ray, FlexRay IP-Module, User's Manual Rev. 1.2.7. Robert Bosch GmbH, 2009, http://www.bosch-semiconductors.de

[15] N.N.: FlexRay Kommunikationscontroller MFR 4310 und Mikrocontroller mit FlexRay-Controller MPC556x und S12XF. www.freescale.com

[16] T. Pop, P. Pop, P. Eles. Z. Pong, A. Andrei: Timing Analysis of the FlexRay Communication Protocol. Real-Time Systems Journal, Springer Verlag, Heft 1-3, 2008, S. 205-235

[17] A. Grzemba (Hrsg.): MOST – The Automotive Multimedia Network. Franzis Verlag, 2. Auflage, 2011 und www.mostcooperation.com

[18] A. Meroth, B. Tolg: Infotainmentsysteme im Kraftfahrzeug. Vieweg+Teubner Verlag, 1. Auflage, 2007

[19] G. Schnell: Bussysteme in der Automatisierungs- und Prozesstechnik. Springer-Vieweg-Verlag, 8. Auflage, 2011

[20] R. Kreifeldt: AVB for Automotive Use White Paper. 2009, AVnu Resource Library, www.avnu.org

[21] R. Stevens: TCP/IP Illustrated. Addison-Wesley, 3 Bände, 2002

[22] E. Hall: Internet Core Protocols. O'Reilly, 2000