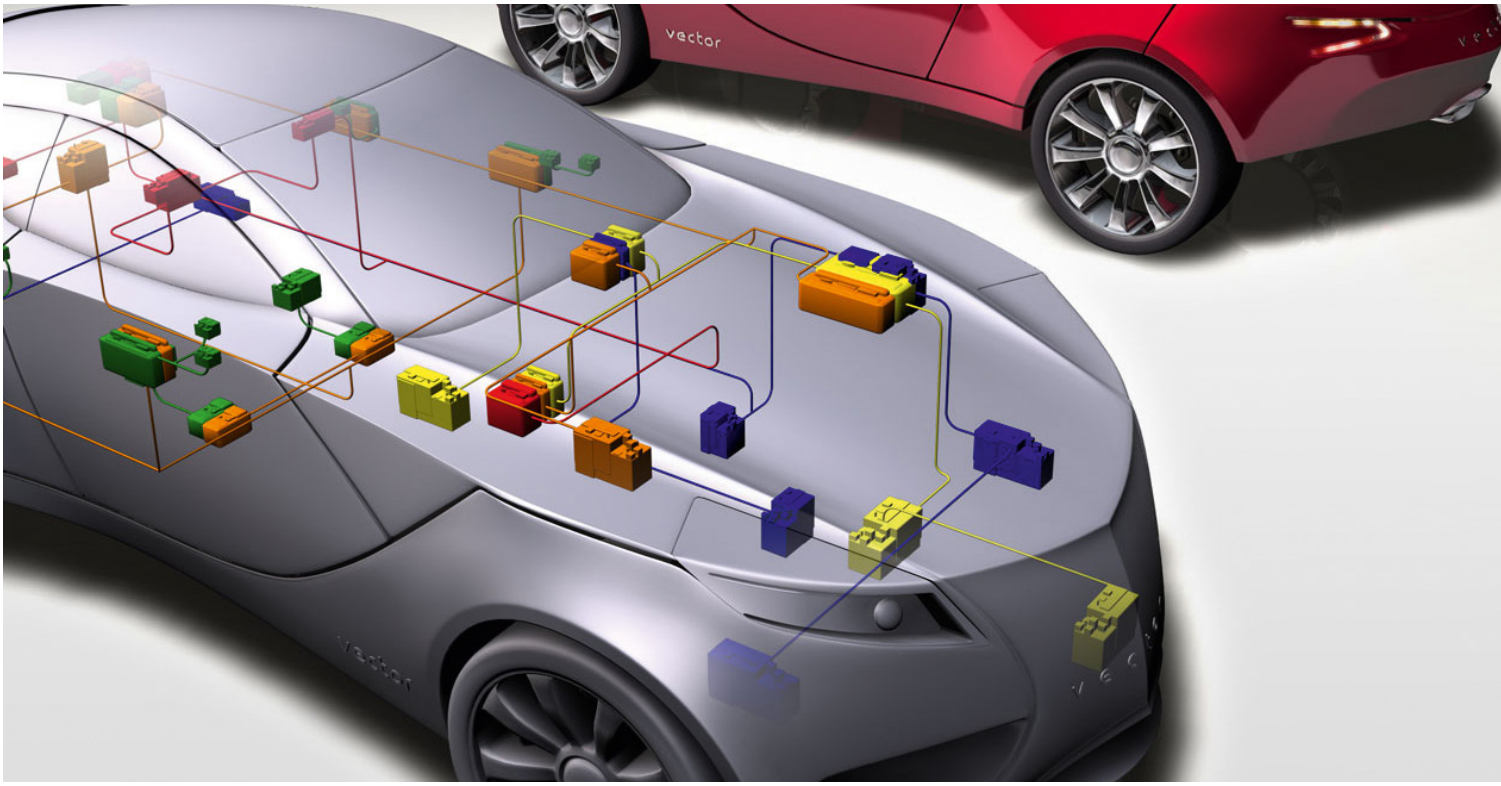

Datenkommunikation im Automobil Teil 3:

Kostengünstiger Datenaustausch mit LIN

Lange Zeit war es üblich, immer größer werdende Zahlen von Sensoren, Aktoren und Motoren direkt mit einem zentralen Steuergerät zu verbinden. Jedoch stieß diese Entwicklung rasch an Grenzen: Der Verkabelungsaufwand stieg, begleitet von größerem Platzbedarf, zunehmendem Gewicht und mehr Störanfälligkeit. Es wurden immer mehr unterschiedliche Kabelbaum- und Steckervarianten nötig, was wiederum die Produktion, Installation und Wartung erschwerte. Man erkannte bald, dass auch in diesem Bereich eine Vernetzung der Komponenten über ein Bussystem ideal sein würde.

Jedoch bietet CAN bei diesen Anwendungen mehr Sicherheit als nötig, wie etwa Sitz-, Fenster-, Spiegel- und Klimasteuerungen. Für solche Komponenten ist der Preis von CAN zu hoch. Hersteller suchten folglich nach billigeren Lösungen. Einfache Busse mit kleinen Übertragungsraten wurden entwickelt. Allerdings kochte jeder Hersteller sein eigenes Süppchen, bis der Preisdruck zu mehr Einheit zwang. Ende 1999 wurde LIN (Local Interconnect Network) geboren. Der folgende Beitrag erklärt den Siegeszug von LIN im Automobil und erläutert seine Funktionsweise.

Inzwischen hat sich diese Technologie auf breiter Front durchgesetzt, so dass man LIN in fast jedem Fahrzeug findet.

Konsortium verhilft LIN zum Durchbruch

Ein wesentlicher Grund für die schnelle Etablierung von LIN war die Gründung des LIN-Konsortiums [1], in dessen Rahmen sich namhafte Kfz-Hersteller und Zulieferer sowie Halbleiter- und Tool-Hersteller zusammengeschlossen hatten, um einen globalen Kommunikationsstandard für den Sensor-/Aktor-Bereich zu schaffen. Mit der Definition eines einfachen und kostengünstigen Physical Layer, der sich an der K-Line (ISO 9141) orientiert, sowie eines einfachen und schlanken Kommunikationsprotokolls, legte das LIN-Konsortium das Fundament für die Realisierung einfacher und kostengünstiger Steuergeräte.

Dabei hatte das LIN-Konsortium nicht nur die eigentliche LIN-Kommunikation im Blick, sondern auch eine einheitliche Entwicklungsmethodik, den sogenannten LIN-Work-Flow. Jedoch setzt sich manchmal selbst gut Gemeintes und gut Geplantes in der Realität nicht durch. Der LIN-Work-Flow ist ein Beispiel dafür.

Zur Beschreibung eines gesamten LIN-Clusters dient eine einheitliche Syntax, die LIN Configuration Language, und das in dieser Syntax verfasste LIN Description File (LDF). Das LDF beschreibt alle Eigenschaften eines LIN-Clusters, besonders die Kommunikation der Steuergeräte. Mit Hilfe

des LDF erzeugen Generierungswerkzeuge die Software-Komponenten für die LIN-Kommunikation. Zusätzlich ist das LDF die Basis für Analyse-, Mess- und Testwerkzeuge oder Restbus-Emulatoren **(Bild 1)**.

Zwei Merkmale unterscheiden LIN von allen anderen Bussystemen. Erstens gibt es zwei Arten von Steuergeräten, nämlich einen LIN-Master pro LIN-Bus und weitere LIN-Slaves, sowie zweitens die Möglichkeit Slave-Steuergeräte in sehr hohen Stückzahlen zu fertigen und als Massenware „von der Stange" an Fahrzeughersteller zu verkaufen. Diese hohen Stückzahlen verringern den Preis weiter und der Markt bietet Herstellern Steuergeräte, für die sie keine eigenen Entwicklungsaufwände betreiben müssen. So finden sich z. B. in jedem Fahrzeug viele kleine Elektromotoren in Sitzen, Spiegeln, Fenstern und Schiebedächern. Und wozu sollte jeder Hersteller zu deren Ansteuerung eigene Steuergeräte entwickeln?

Große Zulieferer bieten solche Slave-Steuergeräte an, deren Fähigkeiten sie mittels einer einheitlichen Syntax, der sogenannten Node Capability Language, im standardisierten Node Capability File (NCF) beschreiben. So beschreibt das NCF Leistungsmerkmale eines LIN-Slaves, wie etwa die Frame- und Signaldefinitionen, Bitraten oder Diagnoseparameter und ist im Rahmen des Systemdesigns die Grundlage zur automatisierten Erstellung eines LDF.

Die detaillierte und übersichtliche Spezifikation für LIN leistete einen weiteren Beitrag zum Erfolg. Die zum Jahreswechsel 2010 auf 2011 vorliegende LIN-Spezifikation 2.2A [2] definiert den Physical Layer, das Kommunikationsprotokoll, den LIN-Work-Flow, die LIN-API sowie die Diagnose und Konfiguration der LIN-Knoten. Seit Januar 2016 liegt der ISO-Standard für LIN vor (ISO 17987). Part 1, 2, 3, 4 und Part 6 werden in den nächsten 2 Monaten publiziert. Part 7 folgt kurz darauf.

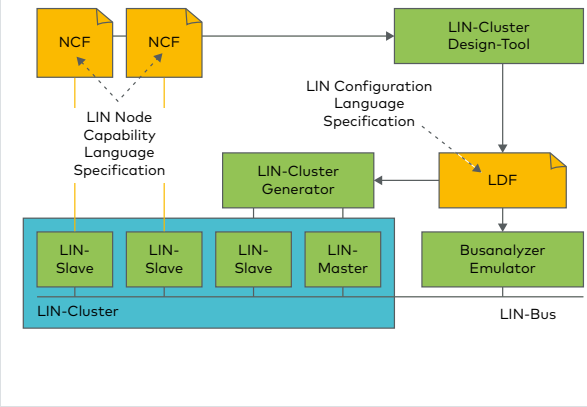

Bild 1: LIN-Cluster

Verzicht auf alles Entbehrliche

Besonders beim Design des Physical Layer ist das Streben nach Kosteneinsparung unübersehbar. Eine gewöhnliche Eindrahtleitung (Single Wire) muss reichen. Auf Spannungsteiler wird verzichtet und man verwendet als Spannungspegel für LIN-Bits die Versorgungsspannung für einen Bit-Wert von 1 und die Fahrzeugmasse für den Bit-Wert von 0. Ein Kommunikationscontroller ist auch nicht erforderlich und als Clock bedarf es nicht immer eines teuren, geeichten Quarzes, sondern ein preiswerter On-Chip-Resonator mit einer Frequenz-Toleranz von bis zu ±14 % tut es auch. In der Praxis ist man jedoch zunehmend von den billigen RC-Clocks abgerückt und LIN-Controller gibt es inzwischen auch.

Ein LIN-Transceiver sorgt für die physikalische Busankopplung. Ein Pegel von weniger als 40 % der Versorgungsspannung wird vom Empfänger als logische Null interpretiert, oberhalb von 60 % der Versorgungsspannung als Eins. Dieser große Unterschied zwischen Null und Eins macht die Bits recht störfest, jedoch sind die Flanken sehr steil, was immer elektromagnetische Störungen zur Folge hat. Um sie gering zu halten wurde die maximale Datenrate auf 20 kBit/s und die maximale Gesamtkapazität auf 2 µF begrenzt. Wegen der Leitungskapazitäten sowie der Kapazitäten der Transceiver kann ein LIN-Bus von 40 m Länge maximal 16 Steuergeräte verbinden.

Wie in der Schule

Ein Cluster besteht aus einem LIN-Master und mindestens einem LIN-Slave. Die Kommunikation wird vom Master geregelt. Software in den Mikrocontrollern übernimmt die Aufgabe des fehlenden Kommunikationscontrollers. Jedes LIN-Steuergerät, auch der Master, verfügt über eine sogenannte Slave-Task. Der LIN-Master besitzt zusätzlich eine Master-Task, mit deren Hilfe er die Kommunikation steuert **(Bild 2)**.

Wie in der Schule darf am LIN-Bus nicht unaufgefordert gesprochen werden. Wie ein Lehrer stellt die Master-Task Fragen, die von der jeweils zuständigen Slave-Task zu beantworten sind. Manchmal stellen Lehrer rhetorische Frage, die sie selbst beantworten. Genauso kann die Mater-Task Fragen stellen, die von der Slave-Task des Masters beantwortet werden.

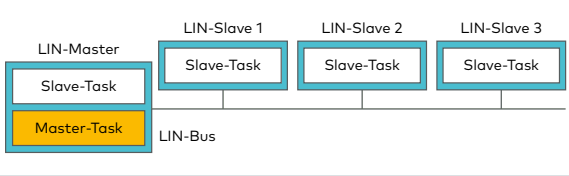

Bild 2: LIN-Master und LIN-Slaves

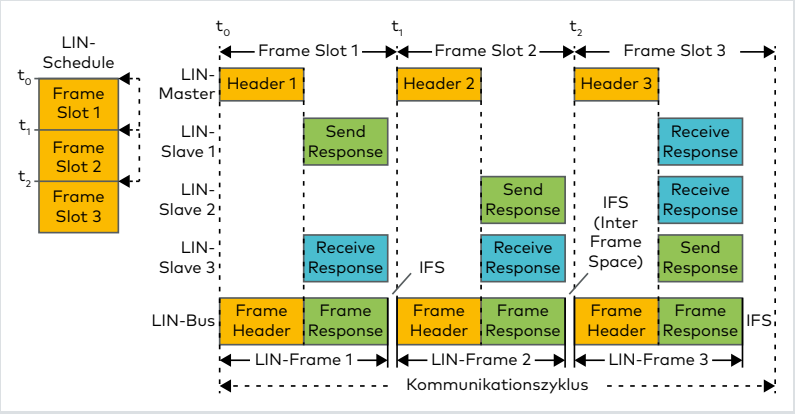

Bild 3: LIN-Kommunikationszyklus

Die zeitliche Abfolge der Fragen erfolgt mittels einer oder mehrerer LIN-Schedules, die in sogenannten Frame Slots organisiert sind. Jeweils zu Beginn eines Frame Slots überträgt die Master-Task einen Frame Header, welcher den Frame-ID beinhaltet. Da dieser ID jedem Slave bekannt ist, wird ein Slave den Header als (An)Frage erkennen und antworten **(Bild 3)**.

Am Identifier erkennen die Slave-Tasks ihre jeweilige Rolle: Eine Slave-Task erkennt, dass sie nun die Daten zum geforderten ID in der sogenannten Frame Response bereitstellen muss, andere sehen, dass sie die nun zu erwartenden Daten auswerten müssen, die übrigen stellen erleichtert fest, dass sie nichts tun müssen.

Frame Header und Frame Response ergeben zusammengenommen einen LIN-Frame. Wie bei CAN steckt im ID eine Nachrichtenadressierung und jedes LIN-Steuergerät darf die Daten lesen (Broadcast).

Datenübertragung mittels LIN-Frames

Weil ein Kommunikationscontroller fehlt, erfolgt die Datenübertragung bei LIN über die serielle Schnittstelle (Serial Communication Interface — SCI) des Mikrocontrollers. Bei CAN wird bitweise gesendet, bei LIN byteweise. Das SCI sendet jedes Byte mit dem Least Significant Bit (LSB) zuerst und rahmt es mit einem dominanten Start-Bit (0) und einem rezessiven Stopp-Bit (1) ein. Man nennt dies SCI-Frame. Ein LIN-Frame setzt sich folglich aus einer Anzahl von SCI-Frames zusammen, die wegen des Start-Bits mit einer fallenden Flanke zur Synchronisation beginnen **(Bild 4)**.

Im Header steckt allerdings mehr als nur der ID. Wegen der billigen, nur ungenau geeichten RC-Clocks können die LIN-Slaves einen ID erst verstehen, nachdem sie einem hinreichend genauen Eichprozess unterzogen wurden.

Erst nach einer Eichung wird LIN verständlich

Zum Eichen überträgt der LIN-Master zunächst eine Sync Break, um alle LIN-Slaves von dessen Beginn in Kenntnis zu setzen. Die Sync Break ist mit mindestens 13 dominanten Bits und einem Stopp-Bit bewusst in der Länge so überdehnt, dass der langsamste LIN-Slave einen SCI-Fehler feststellen muss. Selbst bei 12 dominanten Bits plus ein Stopp-Bit würde ein langsamer LIN-Slave mit der erlaubten Toleranzabweichung von -14 % noch einen ganz „nor-

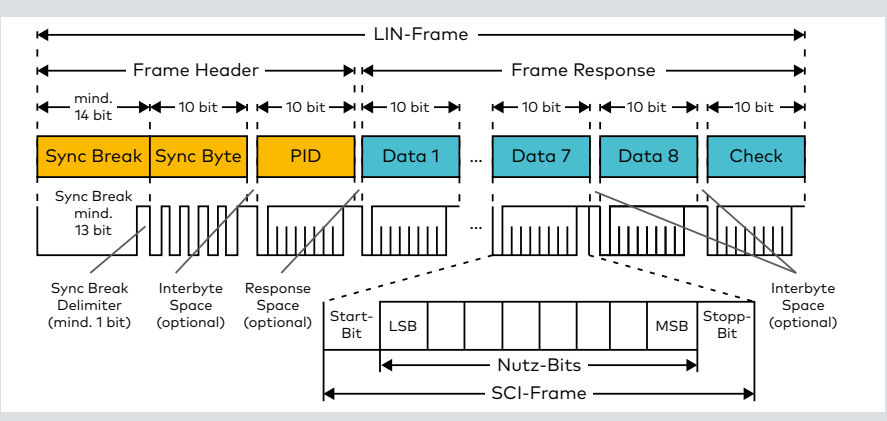

Bild 4: LIN-Frame

malen" SCI-Frame sehen. Jedoch im 13-ten dominanten Bit wird ein Fehler erkannt, weil zu diesem Zeitpunkt ja normalerweise das rezessive Stopp-Bit folgen müsste. Aufgrund dieses Fehlers schaltet der LIN-Slave in einen Modus, um das nun folgende Sync Byte auszumessen. Es enthält den Wert 0x55, also einen Wechsel von Nullen und Einsen, deren fallende Flanken gemessen werden, um die Sendegeschwindigkeit des Masters zu bestimmen. Am Ende sind alle LIN-Slaves mit ±1,5 % Genauigkeit auf den Master geeicht.

Nicht so sicher wie CAN

Der nun folgende ID besteht aus sechs Bit, die durch zwei Paritätsbits gesichert sind. Der ID und die beiden Paritätsbits werden zusammen als Protected Identifier (PID) bezeichnet. Die ersten 60 IDs (0 bis 59) stehen für die Nutzdatenkommunikation zur Verfügung. ID=60 und ID=61 sind für Diagnose. Die letzten zwei Identifier, ID=62 und ID=63, sind reserviert.

Die Frame Response besteht aus mindestens einem und maximal acht Datenbytes, gefolgt von einer acht Bit großen Checksumme für die Fehlererkennung. In den ersten LIN-Versionen wurde die klassische Checksumme eingeführt, ab LIN-Version 2.0 gilt die erweiterte.

Die klassische Checksumme entspricht dem Inversen der Modulo-256-Summe aller Datenbytes. Ein Empfänger bildet seinerseits ebenfalls die Modulo-256-Summe der ankommenden Bytes und addiert als letztes die ankommende Checksumme. Ist das Ergebnis nicht 0xFF, liegt ein Fehler vor. Die erweiterte Checksumme bezieht zusätzlich den PID mit in die Berechnung ein.

Üppige Pausen für langsame Mikrocontroller

Kostengünstige Mikrocontroller sind einfach und oft langsam. Deswegen erlaubt LIN, die Übertragung der Frame Response nach dem LIN-Header ein wenig verzögert zu beginnen (Response Space), und auch Sendepausen (Interbyte Spaces) zwischen den einzelnen SCI-Frames einzulegen. Insgesamt darf sich die Frame Response so um 40 % verlängern. Dasselbe gilt für den Frame Header. Allerdings braucht ein LIN-Master keine Pausen, denn er ist ein Steuergerät, welches in den allermeisten Fällen auch an einem CAN-Bus angeschlossen ist. Die CAN-seitigen Anforderungen erlauben keine Verwendung billiger Mikrocontroller. Der Master nutzt die ihm erlaubten 40 % mehr Zeit auf ein längeres Sync Break: Er halbiert bei dessen Versendung seine Taktrate, sodass 18 dominante Bits plus zwei rezessive Bits herauskommen. Die Zeitreserve von 40 % muss man beim Design der LIN-Schedule unbedingt berücksichtigen.

Vier Typen von Frames

Durch die LIN-Schedule ist ein starres Zeitraster vorgegeben. Jeder Header, der laut Schedule gesendet wird, muss ohne weitere Bedingung mit einer LIN-Response beantwortet werden. Deswegen bezeichnet man solche LIN-Frames als Unconditional Frames.

Jedoch ermöglicht die LIN-Spezifikation, diese Starrheit zu lockern und Frames nur im Bedarfsfall zu senden. Dafür stehen die beiden Frametypen „Sporadic Frame" und „Event Triggered Frame" ab der LIN-Version 2.0 zur Verfügung.

Ein Sporadic Frame ist ein Unconditional Frame, der sich mit anderen Unconditional Frames denselben Frame Slot teilt und vom LIN-Master bedarfsabhängig übertragen wird (Header und Response). Kollisionen sind ausgeschlossen weil nur der Master als einziges Steuergerät solche Frames senden darf. Liegt kein Bedarf beim Master vor, bleibt der entsprechende Frame Slot leer.

Event Triggered Frames kommunizieren sporadische Veränderungen oder Ereignisse auf Seiten der LIN-Slaves. Auch sie entsprechen Unconditional Frames, die sich einen gemeinsamen Frame Slot teilen, doch mit dem Unterschied, dass einem gemeinsamen Frame Header mehrere Frame Responses von unterschiedlichen LIN-Slaves zugeordnet sind. Welcher Slave den Event Triggered Frame Header mit einer Response beantwortet, hängt vom Bedarf der entsprechenden LIN-Slaves ab. Ein Bedarf liegt vor, wenn neue Daten zu übertragen sind. Damit die Empfänger wissen, welche Daten sie erhalten, wird im ersten Byte der Frame Response des Event Triggered Frame der PID des zugrunde liegenden Unconditional Frame mitgeteilt.

Senden mehrere Slaves, folgt eine Kollision für deren Auflösung der LIN-Master verantwortlich ist. Er unterbricht die Schedule nach der Kollision und schiebt eine weitere Schedule ein, in der die Header für alle Event Triggered Frames enthalten sind, die sich den Unglücks-Frame-Slot teilen.

Zur Diagnose von LIN-Slaves stellt LIN zwei reservierte Unconditional Frames zur Verfügung. Sie haben die ID= 60 für den Diagnostic Request und ID=61 für die Diagnostic Response. LIN nutzt das auch bei CAN übliche ISO-TP [3] als Transportprotokoll und einheitliche Diagnosedienste [4, 5].

Status- und Netzwerkmanagement

Die LIN-Spezifikation definiert ein Statusmanagement und ein Netzwerkmanagement.

Das Statusmanagement schreibt den LIN-Slaves ab LIN 2.0 vor, dass sie dem LIN-Master erkannte Übertragungsfehler wie Paritäts- oder Checksummenfehler anzeigen müssen. Dazu bedarf es eines Bits in einem Unconditional Frame, den der Slave an den Master sendet (Status Frame). Weitere Informationen über die Art des Fehlers sind möglich, jedoch keine Pflicht. Auch gibt es keine Anweisungen, wie Fehler behandelt werden. Diese Verantwortung trägt der Entwickler.

Die Aufgabe des LIN-Netzwerkmanagements ist in erster Linie das Ein- und Ausschalten der Slaves, also der Übergang vom normalen Kommunikationszustand (Operational)

in den Sleep Mode und umgekehrt **(Bild 5)**. Erkennen die LIN-Slaves für vier Sekunden keine Busaktivität, dürfen sie vom Operational-Zustand in den Sleep-Zustand wechseln. Bei LIN 2.1 und höher müssen sie dies binnen 10 Sekunden getan haben. Mit einem Sleep-Kommando kann der LIN-Master seine Slaves jederzeit in den Sleep Mode schalten.
Umgekehrt durchläuft ein LIN-Slave eine Initialisierung, um vom Sleep Mode in den Operational-Zustand zu schalten, wenn er ein Wake-Up-Signal gefolgt von einem gültigen Header erkennt. Das Wake-Up-Signal besteht aus einem dominanten Puls von mindestens 250 Mikrosekunden und höchstens 5 Millisekunden Dauer und kann nicht nur vom Master, sondern von jedem LIN-Steuergerät gesendet werden. Die LIN-Spezifikation begrenzt die Initialisierungsphase auf 100 Millisekunden, das heißt, der LIN-Master muss spätestens nach dieser Zeitspanne mit einem Header beginnen. Bleibt er passiv, sendet der entsprechende LIN-Slave ein weiteres Wake-Up-Signal. Abstände und Anzahl an Wiederholungen solcher Signale sind ebenfalls festgelegt.

Fragen Sie die Experten

Unsere Spezialisten von Vector [6] unterstützen Hersteller und Zulieferer der Fahrzeugindustrie und anderen Branchen nicht nur bei der LIN-Vernetzung, sondern auch bei Kommunikationssystemen wie CAN, FlexRay, CAN FD und Ethernet.
Wir bieten durchgängige Ketten aus Werkzeugen für Planung, Design, Entwicklung und Wartung, aber auch Softwarekomponenten und Basissoftware für AUTOSAR-Steuergeräte. Auch für den Einstieg in die Steuergerätevernetzung schulen wir Grundlagen zu CAN, LIN, FlexRay und Ethernet in Seminaren. Wir vermitteln notwendige Fertigkeiten im Umgang mit oben genannten Werkzeugen in Workshops, mit dem Ziel, Ingenieure schnell mit den vielfältigen Entwicklungsaufgaben rund um die Elektronik im Automobil vertraut zu machen [7].
Im ersten Teil dieser Reihe [8] befassten wir uns allgemein mit seriellen Bussystemen, im zweiten Teil mit CAN [9]. In weiteren Folgen werden die seriellen Bussysteme FlexRay und MOST behandelt. Der interessierte Leser findet zu den bereits veröffentlichten Themen auf der Vector E-Learning Plattform [10] ergänzende und vertiefende Informationen.

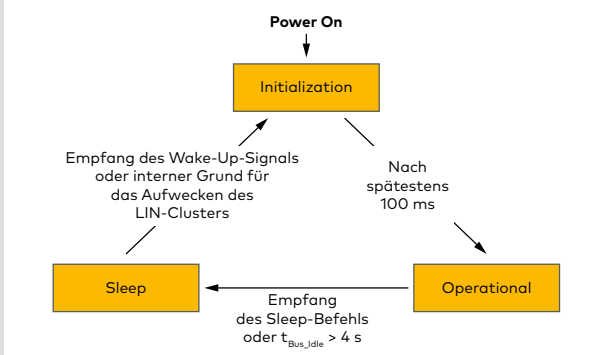

Bild 5: LIN-Netzwerkmanagement

Literatur und Links:

[1] www.lin-subbus.org
[2] LIN Specification Package Revision 2.2A
[3] Road vehicles — Diagnostics on Controller Area Network (CAN) — Part 2: Network layer services. International Standard ISO 15765-2.4, Issue 4, 2002-06-21.
[4] Road vehicles — Diagnostics on controller area network (CAN) — Part 3: Implementation of diagnostic services. International Standard ISO 15765-3.5, Issue 5, 2002-12-12.
[5] Road vehicles — Diagnostic systems — Part 1: Diagnostic services. International Standard ISO 14229-1.6, Issue 6, 2001-02-22.
[6] www.vector.com
[7] www.vector-academy.de
[8] Christmann, E.: Datenkommunikation im Automobil Teil 1: Architektur, Aufgaben und Vorteile.
[9] Christmann, E.: Datenkommunikation im Automobil Teil 2: Sicherer Datenaustausch mit CAN.
[10] elearning.vector.com

Ernst Christmann, Physiker, Mathematiker
ist Senior Technical Trainer im Bereich Software Tools für Steuergerätetests und Steuergeräteentwicklung und seit 2004 bei der Vector Informatik GmbH in Stuttgart tätig.