Lehrstuhl für Realzeit-Computersysteme

Konzept, Realisierung und Bewertung des Sensor/Aktorbusses TTP/A

Robert Huber

Vollständiger Abdruck der von der Fakultät für Elektrotechnik und Informationstechnik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

genehmigten Dissertation.

Vorsitzender: Univ.-Prof. Dr.-Ing. habil. G. Rigoll

Prüfer der Dissertation:

1. Univ.-Prof. Dr.-Ing. G. Färber
2. Univ.-Prof. Dr.-Ing. Dr. h. c. P. Göhner, Universität Stuttgart

Die Dissertation wurde am 12.09.2002 bei der Technischen Universität München eingereicht und durch die Fakultät für Elektrotechnik und Informationstechnik am 25.03.2003 angenommen.

Vorwort

Am 01.01.1999 begann das Forschungsprojekt „Kommunikationsarchitekturen für verlässliche Automatisierungssysteme (Time Triggered Sensor Bus TTSB)“, in dem diese Arbeit entstand. Ziel des dreijährigen Projekts war die Erforschung eines zukunftsorientierten Sensor/Aktorbusses. Der als TTP/A bezeichnete Bus, sollte folgende Attribute besitzen: kostengünstig, realzeitfähig, effizient, skalierbar, offen, sicher, auf Standardbauteilen basierend und einheitlicher Datenzugriff. Obwohl einige der Eigenschaften konträr sind, vor allem gegenüber den Kosten, werden sie in zunehmenden Maße gewünscht bzw. gefordert. Von den existierenden Bussen deckt keiner das Spektrum ganz oder zum größten Teil ab. Deshalb sollte das Forschungsprojekt TTSB auch zeigen, inwieweit das möglich ist.

Projektpartner waren auf der Hochschulseite das Institut für Technische Informatik (Prof. Kopetz) der Technischen Universität Wien, das Institut für Automatisierungs- und Softwaretechnik (Prof. Göhner) der Universität Stuttgart und der Lehrstuhl für Realzeit-Computersysteme (Prof. Färber) der Technischen Universität München. Eine industrielle Beteiligung bestand durch die TTTech AG (Wien), die Softing AG (Haar bei München) und SiemensVDO (Regensburg). Die finanzielle Förderung auf bayerischer Seite übernahm die Bayerische Forschungsstiftung (BFS). Des Weiteren sponserten die Firmen Mentor Graphics und Tasking das Projekt mit kostenfreien Lizenzen für deren „Compiler“, „Debugger“ und weiteren „Tools“ zur Softwareerstellung.

Zwar sind die Forschungsarbeiten noch nicht abgeschlossen. Es zeichnet sich aber bereits ab, dass die Projektziele erreichbar sein werden. Denn die Ergebnisse sind vielversprechend. Grundlage ist ein Konzept, das viele der genannten Eigenschaften vereint und für den Rest die Voraussetzungen schafft. Ein großer Teil von dem Konzept wurde realisiert und mehrfach verifiziert, so dass es mittlerweile ein funktionierendes Basisprotokoll („Basic-TTP/A“) gibt. „Basic-TTP/A“ beruht auf Standardbauteilen, insbesondere auf „low-cost“ µCs mit UARTs, und ermöglicht eine kostengünstige, übertragungseffiziente Umsetzung einfacher und zeitkritischer Sensor/Aktoranwendungen. Ferner konnten Erfolge in puncto Offenheit erzielt werden. Zusammen mit SiemensVDO entstand für „Basic-TTP/A“ ein „High-Speed-Physical-Layer“ für Kfzs. Er soll die momentan höchste Baudrate von 625 kBit/s bis in den MBits/s-Bereich steigern.

Mit Hilfe der Universität Stuttgart war es möglich, die Skalierbarkeit zu beweisen und erste Schritte in Richtung eines einheitlichen Datenzugriffs zu machen. Ein „IFS/XML-Gateway“, das auf „Basic-TTP/A“ aufsetzt, ermöglicht den Anschluss eines Demonetzes an einen PC mit „Web-Server“. Auf dem PC befindet sich eine „Web-Site“, mit der man das Demonetz über das Internet in einem „Web-Browser“ beobachten und fernsteuern kann; dieser Versuch wurde unter anderem auf der Messe „Interkama 2001“ in Düsseldorf gezeigt. Das Ergebnis der Kooperation mit der Technischen Universität Wien ist die TTP/A-Spezifikation und ein Standardisierungsvorschlag bei der OMG („Object Management Group“). Außerdem trug die Technischen Universität Wien, mit ihrer Erfahrung auf dem Gebiet der „Time-Triggered Architectures“ (TTA), zum Gelingen des Forschungsprojekts TTSB bei.

An dieser Stelle möchte ich mich bei allen Beteiligten und Sponsoren bedanken, vor allem bei Herrn Prof. Färber für die Unterstützung während dieser Arbeit und deren Begutachtung und Herrn Prof. Göhner für die Begutachtung dieser Arbeit. Des Weiteren möchte ich ein ganz besonderes Dankeschön an Dipl.-Ing. Bernd Pfaffeneder von SiemensVDO und an Dipl.-Ing. Stephan Eberle von der Universität Stuttgart richten. Aufgrund der offenen, angenehmen und effizienten Zusammenarbeit, entstand im Laufe des Projekts eine Freundschaft zwischen uns, die ich sehr schätze. Zudem bedanke ich mich bei meinem „Zimmergenossen“ Dipl.-Ing. Gregor Burmberger, der mit seiner Gabe Fehler aufzuspüren zur Stabilität der Implementierung beigetragen hat. Und schließlich möchte ich ein herzliches Dankeschön an meinen Vater und an meinen guten Freund Dipl.-Ing. Elmar Krammer, für das sehr sorgfältige Korrekturlesen und die vielen Tipps, aussprechen.

Abschließend ein paar Anmerkungen: Grundlagen werden in dieser Arbeit nicht erörtert, sondern an Ort und Stelle nur erwähnt, sofern sie für das Verständnis wichtig sind. Fremdsprachliche Fachbegriffe (z.B. „Compiler“) und Kunstwörter (z.B. „Basic-TTP/A“) stehen in Anführungszeichen. Hervorhebungen und Anmerkungen sind *kursiv* oder unterstrichen dargestellt. Außerdem ist der Text in der neuen deutschen Rechtschreibung verfasst.

Inhalt

1 Einleitung

1.1 Motivation für diese Arbeit

Entwicklungen

Seit ein paar Jahren berichten die Fachmedien verstärkt über Bussysteme für sicherheitskritische Anwendungen. An vorderster Stelle steht die Kfz-Technik mit dem mittlerweile gängigen Begriff „Drive-by-Wire“. Gleich danach folgt die Automatisierungstechnik mit busbasierten Schutzeinrichtungen, wie Notaussysteme oder Lichtgitter. Nun sind Bussysteme ja nichts Neues, es gibt sie seit über 30 Jahren. Auch ihr Einsatz in sicherheitskritischen oder verlässlichen Anwendungen ist nicht neu. In der Luft- und Raumfahrt oder im Militär arbeitet man auf diesem Gebiet seit mehr als 10 Jahren, z.B. mit dem SAFEbus oder ARINC 627. Trotzdem sorgt dieses Thema für zahlreiche Entwicklungs- und Forschungsaktivitäten. Eine Recherche im Rahmen dieser Arbeit hat gezeigt, dass etwa 20 neue Bussysteme entstehen bzw. kürzlich entstanden. An den Arbeiten sind bzw. waren viele namhafte Automobilhersteller, Zulieferer, Chiphersteller und einige Universitäten beteiligt. Die Hälfte der neuen Busse betreffen sicherheitskritische Anwendungen auf Geräte- bzw. Feldebene[1]. Dabei spielen nicht nur die wirtschaftsstarken Sparten Automatisierungs- und Kfz-Technik eine Rolle, sondern auch die Bahntechnik und die Luft- und Raumfahrt. Von diesen Bussystemen sind jedoch nur wenige wirklich neu (Byteflight, TTP/C, FlexRay, Spider). Die meisten sind Varianten etablierter Feldbusse.

Das ist bei der zweiten Hälfte anders[2]. Hier handelt es sich fast ausschließlich um Neuentwicklungen und laufende Projekte. Aus diesem Grund ist diese Gruppe, zu der auch TTP/A gehört, weniger bekannt. Der Focus ist auf die Sensor/Aktorebene gerichtet. Als Einsatzgebiet sind einfache und anspruchsvolle Anwendungen aus den genannten Sparten, der Haustechnik, Medizintechnik und weiteren Bereichen geplant. Von allen ist die Kfz-Technik am interessantesten. Sie weist momentan das größte wirtschaftliche Potenzial auf, weshalb viele der neuen Sensor/Aktorbusse darauf ausgerichtet sind. Zu den einfachen Kfz-Anwendungen zählen die Vernetzung von Fensterhebern, Sitzheizungen, Temperaturfühlern, Regensensoren, Lampen usw. Anspruchsvoll bzw. sicherheitskritisch sind hingegen „Crash“-Sensoren, „Airbags“, Gurtstraffer und Überrollbügel. Eine besondere Herausforderung im Sensor/Aktorbereich ist die scharfe Kostenrestriktion. Bis heute gibt es keinen Bus, der für die Verschmelzung von einem einfachen Sensor und Aktor mit der Buselektronik günstig genug ist. Mit den neuen Sensor/Aktorbussen soll diese Hürde genommen werden. In der hier tonangebenden Kfz-Technik wird die Schranke für einfache Sensor/Aktoranwendungen bei ungefähr 0,5..1 € pro Teilnehmer gesetzt und für die sicherheitskritischen bei 1..2 € [71]. Das Kostenproblem versuchen die meisten

---

[1] Byteflight, FlexRay, TTP/C, TTCAN, Interbus-Safety, ProfiSafe, Safety-at-Work, SafetyBus (Pilz), DeviceNet-Safety, JetWeb, TCN, Spider

[2] Bluetooth, LIN, TTP/A, PPC, BST, DSI, DCBus, SURFS, BoTe, Planet, SafetyBus (Delphy), UART-Token

der neuen Sensor/Aktorbusse über ASICs, sehr hohe Stückzahlen und eine Spezialisierung zu lösen. Lediglich der LIN- und TTP/A-Bus setzen Standardbauteile ein. Beide Anwendungsgebiete deckt hingegen nur der TTP/A-Bus konzeptionell ab.

Systemkomplexität

Die Fakten machen den Bedarf an Sensor/Aktorbussen offensichtlich. Allerdings erklären sie nicht, wie er zu Stande kam. Denn bisher wurden die Sensoren und Aktoren mit Steuergeräten, SPSen, IPCs oder Anschlussmodulen in der Regel direkt verdrahtet. Aus diesem Grund gibt es auch wesentlich mehr Feld- als Sensor/Aktorbusse. Der ASI-Bus ist momentan der einzige Sensor/Aktorbus, wenn man von proprietären und veralteten Modellen absieht. Er ist etabliert, standardisiert und besitzt einen beachtlichen Marktanteil. Laut einer Studie [25] über den Einsatz von Bussystemen in mehr als 300 großen europäischen Fabriken, war der ASI-Bus Anfang 1999 mit etwa 7 % vertreten[3]. Mit Kosten in der Größenordnung von 5..10 € pro „Chip“ [4] rentiert er sich allerdings nur in großen, hochautomatisierten Industriebetrieben. Dort kostet jede Minute Produktionsstillstand oder eine verzögerte Inbetriebnahme sehr viel Geld. Und da der ASI-Bus einen Zeitgewinn bei der Installation und Wartung bietet, amortisiert er sich in großen, hochautomatisierten Industriebetrieben schnell. Für andere Sparten ist diese Kostenhürde zu hoch, obwohl sie von der Zeitersparnis gerne profitieren möchten. Denn durch die steigende Komplexität werden die Maschinen, Geräte und Systeme immer aufwändiger in der Produktion und Wartung.

Verantwortlich hierfür zeichnen technische Innovationen in der Rechnertechnik. Mit der zunehmenden Rechenleistung von µPs, µCs oder DSPs steigt die Funktionalität und Komplexität der implementierbaren Algorithmen. Dadurch lassen sich technische Prozesse optimieren, verstärkt automatisieren und neue Aufgabengebiete erschließen. Roboterfabriken sind schon fast alltäglich. Ebenso Autos mit Fahrerunterstützung beim Bremsen, Abstandhalten und Einparken. In der Medizintechnik wurden erste Versuche mit aktorischen Prothesen gemacht, z.B. Beinprothesen mit Gehunterstützung. Zukünftig sollen „Airbags“ intelligent gezündet werden, um die Verletzungsrisiken bei einem Autounfall weiter zu reduzieren. Bald werden auch teilautonome Roboter schwierige Operationen am Menschen durchführen und vollautonome Putzroboter die lästige Hausarbeit übernehmen. Dieser ständige Fortschritt hat für die Wirtschaft eine große Bedeutung, weil er Märkte sichert und neue erobert.

Mit ihm wächst jedoch die Systemkomplexität und zwar auch auf der Sensor/Aktorebene. Denn intelligente und autonome Maschinen müssen ihre Umwelt bzw. ihren technischen Prozess verstärkt wahrnehmen und beeinflussen können. Aus diesem Grund stieg und steigt die Anzahl und Vielfalt von Sensoren und Aktoren pro System ständig an. Neue Gesetze tragen dazu ebenfalls bei. Verschärfte Abgasvorschriften für Kfzs, Kohlekraftwerke und Müllverbrennungsanlagen haben zu einer aufwändigeren Mess- und Steuerungstechnik geführt. Ähnliches ist beim Energieverbrauch zu erwarten, wegen den knapper werdenden Ressourcen und der CO_2-Misere. Heute ist das 3 l-Auto oder das intelligente Niedrigenergiehaus zwar noch kein Muss, ein Trend dorthin zeichnet sich aber ab. Die gestiegene Produkthaftung durch die CE-Vorschriften liefert weitere Beispiele. Eines davon sind Fenstergummilippen mit Drucksensoren, die Verletzungen durch automatische Fensterheber verhindern sollen – und damit Regressansprüche.

Produktion und Wartung

Die Produktion und Reparatur dieser Systeme werden allerdings immer schwieriger. Mit der Anzahl der Sensoren und Aktoren werden die Kabelbäume dicker und es steigt die Anzahl der Verbindungs-

[3] Profibus 39 %, Interbus-S 13 %, Ethernet 13 %, Rest 28 %.

[4] Der höhere Preis betrifft kleine, der niedrigere Preis große Abnahmemenge – Letzterer wurde geschätzt.

stellen. Damit nehmen die Verdrahtungs- und Kontaktfehler zu. Schaltschränke oder Gebäudeinstallationen z.B. werden nach wie vor in mühevoller und zeitaufwändiger Handarbeit angefertigt. Trotz äußerster Sorgfalt passieren Klemmfehler, in Form von vertauschten oder schlecht geklemmten Drähten. Statistisch gesehen nehmen sie mit der Anzahl der Drähte zu und damit auch mit den verbauten Sensoren und Aktoren. In einem Kfz ist es ähnlich. Zwar werden dort codierte Stecker verwendet, doch jemand muss auch sie konfektionieren. Sensor/Aktorbusse können hier Abhilfe schaffen. Sie reduzieren die Kabelmenge und vereinfachen die Klemmpläne. Außerdem erleichtern sie die Verdrahtung, weil jeder Sensor und Aktor auf die gleiche Weise angeschlossen wird. Die Vorteile liegen auf der Hand: weniger Fehlerquellen, bessere Übersicht und kürzere Fertigungs- bzw. Installationszeiten. Beim ASI-Bus z.B. „schnappt“ man einen Teilnehmer einfach auf das Buskabel. Des Weiteren können Sensor/Aktorbusse die Inbetriebnahme beschleunigen, sofern sie „Plug&Play“ unterstützen, weil damit zeitraubende und fehleranfällige Konfigurationsphasen entfallen.

Bei der Reparatur und Wartung sind Sensor/Aktorbusse ebenfalls nützlich. Defekte an Kabeln lassen sich, wegen deren deutlich geringeren Anzahl, leichter lokalisieren. Gleiches gilt für die schwer auffindbaren Kontaktfehler. Hier helfen intelligente Diagnosetechniken, die einen Rückschluss auf den Fehlerort zulassen (z.B. über die Teilnehmerfehlerhäufigkeit). Mit einer intelligenten Diagnose können auch Fehler in den Sensoren und Aktoren erkannt werden. Aufgrund physikalischer Gesetzmäßigkeiten dürfen sich Sensorwerte nur in einem definierten Bereich befinden oder sich mit einer bestimmten Geschwindigkeit verändern. Eine Überschreitung von derartigen Grenzwerten kann ein Hinweis auf einen internen Defekt oder einen sich anbahnenden Ausfall sein. Intelligente Sensoren vermögen diese Situation, mit relativ einfachen Plausibilitätstests, zu erkennen und zu melden; aufgrund von Übertragungsfehlern kann das ein zentraler Steuerrechner nur bedingt. Ein weiterer Vorteil ist die einfachere, zielgenauere, präventive Wartung. Mit geringem Aufwand (interner Betriebsstundenzähler) können intelligente Sensoren und Aktoren den Ablauf ihrer MTBF rechtzeitig melden. Wenn sie zudem „hot-plugable“ sind, lassen sich kleinere Reparatur- und Wartungsarbeiten während des Betriebs erledigen.

Sicherheit und Verlässlichkeit

Daneben erhöhen Sensor/Aktorbusse die Sicherheit und Verlässlichkeit. Und zwar nicht nur wegen der geringen Fehlerquellen, sondern auch wegen der besseren elektromagnetischen Verträglichkeit (EMV). Da jeder Teilnehmer an einem Sensor/Aktorbus einen Bustreiber besitzt, werden Mess- und Stellwerte stets niederohmig übertragen. Außerdem vergrößert die Vorort-Digitalisierung bei analogen Signalen den Störspannungsabstand. Und schließlich ermöglicht eine Codierung den besseren Umgang mit Übertragungsfehlern. Diese Eigenschaften machen die Signalübertragung mit einem Sensor/Aktorbus störfester als mit der konventionellen Verdrahtungstechnik. Denn dort werden hochohmige Signale oftmals erst nach mehreren Metern Kabel an einen Steuerrechner angeschlossen. Sensoren sind hiervon stärker betroffen als Aktoren und analoge Signale stärker als binäre. Bei der Störabstrahlung und Eigenfestigkeit haben Sensor/Aktorbusse einen weiteren Vorteil. Aktoren steuern elektrische Leistung zur Verrichtung von (Stell-)Arbeit. Dabei breiten sich Felder von den leistungsführenden Drähten aus, die benachbarte Schaltkreise stören können. Wegen der verlustarmen PWM-Technik, haben die Störungen zugenommen. Je schneller man den Leistungsfluss ändert, desto stärker werden die Störfelder. Für lange leistungsführende Anschlussdrähte gilt dasselbe. Auf den zeitlichen Leistungsfluss hat man keinen Einfluss, jedoch auf die Drahtlänge. Bei einem Sensor/Aktorbus sind die Anschlussdrähte meistens kurz, weil der benötigte Verstärker i.A. Teil des Aktors ist. Dagegen sind sie bei der konventionellen Verdrahtungstechnik eher lang. Hier sind Verstärker und Aktor häufig getrennt. Ein gutes Beispiel sind die elektrischen Motoren mit integriertem Schaltverstärker und Bus-

anschluss („Smart-Drives“). Ihre elektromagnetische Verträglichkeit ist relativ harmlos. Diskret aufgebaut, also in konventioneller Technik, zählen sie zu den problematischsten Störquellen.

Kostendilemma

Sensor/Aktorbusse bieten also eine Menge Vorteile. Aufgrund des beschriebenen Verhältnisses zwischen Anschaffungskosten und „Return of Investment“, können davon bisher nur große Industriebetriebe profitieren. Bei Autos, Hausinstallationen, medizinischen Prothesen, Operations- und Putzrobotern etc. stehen jedoch die Anschaffungs- oder Produktionskosten im Vordergrund. Bis heute verdrängen sie den offensichtlichen Bedarf an Sensor/Aktorbussen. Glücklicherweise wird dieses Dilemma mit den neuen Sensor/Aktorbussen ein Ende haben.

1.2 Warum noch ein Bus?

Wenige relevante Bussysteme

Angesichts über 50 existierender Feldbusse weltweit [12] [16] und etwa 20 neuer Feld- und Sensor/Aktorbusse, ist diese Frage gerechtfertigt. Von den existierenden Feldbussen sind lange nicht alle relevant. Viele Feldbusse, wie z.B. der über 18 Jahre alte Bitbus von Intel oder die SLIO-Variante von CAN, sind obsolet. Einige der Feldbusse sind auch proprietär und erleiden dasselbe Schicksal, wegen einer Tendenz hin zu standardisierten Bussystemen. Beides zusammen reduziert die Anzahl auf etwa 15..20 relevante Feldbusse. Von den neuen Bussystemen werden auch nicht alle überleben. Der TTP/C-, FlexRay-, Byteflight- und TTCAN-Bus konkurrieren um den „Drive-by-Wire“-Einsatz. Ein oder zwei Busse reichen hierfür aus. Gleiches gilt für die neuen Sensor/Aktorbusse. Der PPC-Bus macht z.B. dem LIN-Bus Konkurrenz und der Planet-Bus z.B. dem BST-Bus.

Außerdem gelingt auch nicht jedes neue Bussystem. Der erste Versuch des Planet-Busses scheiterte an der Kostenhürde. Für eine potenzialfreie, differentielle Übertragung setzte Philips kleine Transformatoren ein. Diese kostspielige Lösung führte zu einer kompletten Überarbeitung; jetzt verwendet Philips einen Gleichrichter mit zwei Kondensatoren. Ein anderes Beispiel ist der DSI-Bus von den Firmen TRW und Motorola. Mit ihm sollten „Airbags“ und andere Unfallschutzeinrichtungen gesteuert werden. Vermutlich wegen Sicherheitsbedenken, wurde er nur kurze Zeit eingesetzt. Gemäß der Spezifikation 1.0 [64] arbeitet der DSI-Bus mit einer 4 Bit CRC-Sicherung für Telegramme mit bis zu 16 Datenbits. Überschlägig erreicht er damit eine Hamming-Distanz von max. drei; minimal sind jedoch vier gefordert [104]. Wegen der Telegrammlänge ist zudem die Restfehlerwahrscheinlichkeit relativ hoch. Außerdem wurde das Generatorpolynom $g(x) = x^4 + 1$ ungünstig gewählt. Es kann in vier gleiche Primitivpolynome ($(x + 1)^4$) zerlegt werden, wodurch die Hamming-Distanz, selbst bei nur einem Datenbit, nicht über zwei hinausgeht.

Realzeitprobleme

Trotzdem bleiben noch genügend Bussysteme übrig. Warum also mit TTP/A noch einen Bus schaffen? Der beschriebene Kostenaspekt, dem TTP/A, im Gegensatz zu den meisten anderen Bussen, mit Standardbauteilen Rechnung trägt, ist ein Grund. Er ist gewichtig, aber nicht der Einzige. Für sichere und verlässliche Systeme ist die Realzeitfähigkeit eine notwendige Voraussetzung [103] [136]. Viele Bussysteme haben damit ein Problem. Es entsteht durch die automatische Wiederholung verfälschter Telegramme. Unabhängig von der Realzeitstrategie, kann die Wiederholung ein nachfolgendes Telegramm soweit verzögern, dass es zu einer Realzeitverletzung kommt. Es besteht sogar die Möglichkeit einer Fehlerfortpflanzung, wenn verzögerte Telegramme selbst zur Ursache von Verzögerungen werden. Für Telegrammwiederholungen Platz zu lassen ist unpraktikabel, weil man auch Mehrfachwie-

derholungen berücksichtigen müsste. Zwar passieren diese nicht beliebig oft, damit „Bubbling-Idiots“ den Busverkehr nicht dauerhaft stören können, aber oft genug, um die Busauslastung in die Tiefe und die Latenzzeiten in Höhe zu treiben. Deshalb gehören automatische Wiederholungen nicht in Realzeit-Bussysteme. Ihre Stärken liegen bei der komfortablen Sicherung realzeitunkritischer Daten.

Leider hat man auf die Wiederholmechanismen keinen Einfluss. Erstens sind sie für den Anwender nicht zugänglich, zweitens tief in den Protokollen verwurzelt und drittens wegen der gängigen monolithischen Architektur schwer änderbar. Trotzdem hat man z.B. beim CAN-Bus über eine Änderung nachgedacht. Geplant ist eine Busvariante für realzeitkritische Anwendungen namens TTCAN. Die notwendigen Änderungen sind jedoch so schwerwiegend, dass der TTCAN-Bus mit dem ursprünglichen Protokoll nicht mehr kompatibel ist [29]. Vor diesem Hintergrund ist der CAN-basierte Safety-Bus von Pilz, der realzeitkritische Daten mit periodischen Nachrichten höchster Priorität zu übermitteln versucht, kritisch zu betrachten. Ein weiteres Beispiel ist das Ethernet. Aufgrund der CSMA/CD-Arbitirierung sind Telegrammwiederholungen hier fundamental. Deshalb arbeitet das Ethernet heute vorwiegend mit Punkt-zu-Punkt-Verbindungen über „Switches“. Damit lassen sich zwar Telegrammkollisionen vermeiden, jedoch nicht die Wiederholungen im Fehlerfall. Aus diesem Grund muss auch das darauf basierende JetWeb kritisch betrachtet werden. Viele andere Bussysteme (ASI, Profibus...) sind von dem Wiederholungsproblem ebenfalls betroffen. Der Realzeitaspekt stand bei den Bussystemen früher nicht so im Vordergrund wie heute.

Ein weiterer Punkt ist die Realisierung der Realzeitsteuerung. Bei zahlreichen neuen und modifizierten Bussen (LIN, BST, ProfiSafe, SafetyBus...) befindet sich die Realzeitsteuerung in der Applikation. Dadurch besteht die Gefahr der Beeinflussung des Realzeitverhaltens durch die Applikation, insbesondere bei Änderungen. Ein funktionierendes System wird nach kleinen Anpassungen häufig nur oberflächlich getestet. Denn der Mensch denkt nun einmal linear: kleine Änderung, kleine Auswirkung, kleiner Test. Leider trifft das in Rechnersystemen selten zu. Hier können scheinbar kleine Änderungen versteckte, sprunghafte Auswirkungen haben (Prioritätsinversion, „Dead-Locks“...). Eine implizite Realzeitsteuerung verhindert derartige Interferenzen. Entsprechende Bussysteme gibt es jedoch nur wenige: ASI, TTP/C, Byteflight, FlexRay und TTP/A.

Übertragungseffizienz

Außerdem spielt die Übertragungseffizienz (kurz Effizienz) eine Rolle. In der Sensor/Aktorebene kommen hauptsächlich kleine Datenpakete vor. Für binäre Sensoren und Aktoren, die in der Automatisierungstechnik mit 80..90 % vertreten sind [45], reicht prinzipiell ein Bit aus. Danach folgen 8 Bit-Werte für z.B. Temperatur, Druck oder Helligkeit. Etwas seltener sind 12 Bit- und selten 16 Bit-Werte. Natürlich sind auch Multisensoren und/oder Aktoren möglich, z.B. in Form von Anschlussmodulen. Allerdings macht das in wenigen Fällen Sinn. Erstens wegen des Kabelproblems und zweitens wegen der intelligenten Anwenderunterstützung. Das heißt, dass man mit einem Datenbyte ungefähr 90 % aller Anwendungen abdecken kann. Feldbusse hingegen sind für größere Datenmengen ausgelegt. Beispiel Profibus: Hier beginnen die Pakete mit acht und enden bei 246 Datenbytes. Für eine vernünftige Fehlersicherung, muss die Anzahl der Prüfbits entsprechend hoch sein. Sie variiert beim Profibus mit den Datenbytes, beträgt aber mindestens 16 (Hamming-Distanz = 4). Alles zusammen macht die Übertragung von Sensor/Aktordaten mit einem Feldbus ineffizient. Ein Sensor/Aktorbus berücksichtigt diesen Aspekt stärker. Natürlich nur für Sensor/Aktordaten, denn mit zunehmender Datenmenge werden die Sensor/Aktorbusse in der Effizienz schlechter, wohingegen die Feldbusse besser werden.

Baudrate und EMV

Die Folgen einer ineffizienten Übertragung sind zwar nicht dramatisch, aber auch nicht von der Hand zu weisen. Bei gleicher Übertragungszeit wird die Baudrate umso höher, je ineffizienter die Übertragung ist. Der Faktor kann durchaus zehn und mehr betragen wie die folgende Abschätzung zeigt: Mit acht Datenbits liegt die Telegrammeffizienz beim TTP/A-Bus bei etwa 55 % (siehe 4.3.2), beim ASI-Bus bei 28,6 % (siehe 2.3.1) und beim LIN-Bus bei 18 % (siehe 2.3.3). Dagegen schafft der Profibus[5] nur ca. 5,2 % und das Ethernet[6] 1,4 %. Ohne Berücksichtigung von weiteren Protokolldetails (Pausen, Quittierungen, Baudraten...), müsste das Ethernet mit ca. 3,9 MBit/s und der Profibus mit ca. 1 MBit/s arbeiten, um mit dem LIN-Bus bei ca. 306 kBit/s oder dem ASI-Bus bei ca. 192 kBit/s mitzuhalten. Für den TTP/A-Bus würden dagegen 100 kBit/s ausreichen. Nun ist die EMV eines Bussystems umso besser, je niedriger die Baudrate ist. Und zwar bezüglich der Störaussendung und Störempfindlichkeit. Für die Störaussendung spielen bei den Sensor/Aktorbussen hauptsächlich die Signalflanken eine Rolle (unmodulierte Übertragung). Sie müssen mit zunehmender Baudrate steiler werden, damit die rechteckähnliche Signalform möglichst erhalten bleibt. Leider wächst das Frequenzspektrum damit überproportional an, wodurch die Harmonischen entsprechend stärker abstrahlen[7]. Außerdem nehmen die Reflexionen an Störstellen im Ausbreitungsmedium zu. Davon ist in erster Linie die Kfz-Technik betroffen. Wegen der Kabelbaum- und Kostenproblematik, muss man dort mit Stichleitungen arbeiten, die einige Meter lang sind. Beim CAN-Bus werden diese Stichleitungen offen gelassen, weil der Standard nur zwei Abschlusswiderstände vorsieht. Die Folge ist eine Baudratenbegrenzung bei 500 kBit/s für einen unproblematischen Betrieb.

Wichtiger als die Störaussendung ist die Störempfindlichkeit. Ein störarmer Bus ist wünschenswert, ein störfester aber notwendig. Denn ein Bus der z.B. bei jedem Handy-Anruf zusammenbricht kann weder sicher noch verlässlich sein. Effiziente Bussysteme haben diesbezüglich einen Vorteil, weil die Störempfindlichkeit umso schlechter wird, je höher die Baudrate ist. Diese praktische Erfahrung kann man mit dem Shannon'schen Satz der Informationstheorie tendenziell belegen. Shannon hat bewiesen, dass die max. Baudrate eines Übertragungskanals (C) von der Bandbreite (B), Signalleistung (S) und Rauschleistung (R) wie folgt abhängt: $C = B \cdot ld(1+S/R) \cdot bit$. Durch Umstellen erhält man eine Formel, die das min. Signal-Rauschleistungsverhältnis für eine gewünschte Übertragungsgeschwindigkeit, bei einer gegebenen Bandbreite, wiedergibt: $S/R = 2^{C/B} - 1$. Hieraus kann man ableiten, dass das erforderliche Signal-Rauschleistungsverhältnis exponentiell mit der Baudrate zunimmt, sofern der Kanal voll ausgenutzt wird, die Bandbreite konstant und das Rauschen Gauß-verteilt ist. Diese Betrachtung ist rein theoretischer Natur, weil Störsignale häufig gebündelt auftreten und die max. Kanalkapazität nie genutzt wird[8]. Trotzdem hat die Formel eine gewisse praktische Bedeutung, indem sie die Richtung weist. Bei gleicher Nettodatenrate und Busphysik lässt sich ein effizienter Bus wesentlich weniger stören als ein ineffizienter. Wiegen hingegen die Kosten mehr, so kann man einen effizienten Bus, gegenüber einem ineffizienten, bei gleicher Störempfindlichkeit günstiger realisieren (z.B. Ein- oder nichtverdrillte Zweidrahtleitung, keine Schirmung...). Aufgrund seiner relativ hohen Effizienz, hat der TTP/A-Bus hier deutliche Vorteile gegenüber vielen anderen Bussen.

---

[5] Das kürzeste Profibus-Telegramm mit Daten umfasst 14 UART-„Frames" à elf Bits.

[6] Heutige Ethernet-Varianten benötigen min. 72 Bytes lange Telegramme, damit die Teilnehmer die CSMA-typischen Buskollisionen erkennen [16]. Falls zu wenige Daten vorliegen, fügen die Ethernet-„Chips" Leerbytes ein („Pad"-Feld).

[7] Bei einem Trapezimpuls ist der 40 dB/Dekade-Knick vorteilhaft. Er wandert mit $1/T_{Flanke}$ durch das Frequenzspektrum.

[8] Ein gutes Kabel kann man im Meterbereich durchaus mit 100 MBit/s betreiben [100]. 100Base-TX mit Kategorie-5-Kabeln ist ein Beispiel dafür.

Rückwirkungsfreiheit

Für TTP/A spricht zudem die Rückwirkungsfreiheit. Über einen Bus werden in der Regel mehrere unabhängige, technische Prozesse gesteuert, wodurch die Möglichkeit der gegenseitigen Beeinflussung besteht. Wenn man einmal von einem „Bubbling-Idiot“ und anderen Extremfällen absieht, kann das auch bei einfachen Dingen, wie einem Teilnehmerausfall oder Wartungsarbeiten im Betrieb, passieren. Der ASI-Bus z.B. „pollt“ nur vorhandene, intakte Teilnehmer. Sollte ein Teilnehmer ausfallen oder abgesteckt werden, dann entfernt er diesen, nach ein paar Fehlversuchen, aus dem Abfragezyklus. Umgekehrt verhält es sich genauso – bis auf die Fehlversuche natürlich. Der ASI-Bus erkennt neue Teilnehmer und integriert sie selbstständig in dem Abfragezyklus. Diese Anwenderfreundlichkeit hat jedoch einen Nachteil. Sie verkoppelt unabhängige, technische Prozesse, indem sie die Abtastzeiten aller Teilnehmer variiert. In einfachen Anwendungen hat das keine Konsequenz. Bei einer Regelung, die über den Bus geschlossenen ist, macht sich das jedoch bemerkbar. Wenn die Abtastzeit vom Buszyklus abhängt und die Reglerparameter konstant sind, so ändert sich das Regelkreisverhalten mit der Anzahl der Teilnehmer. Sollte die Abtastzeit hingegen unabhängig sein, entsteht ein großer Abtast-„Jitter“[9] mit entsprechenden Folgen für die Totzeit, aufgrund der Asynchronität. Beide Szenarien beeinflussen die Regelkreisgüte und können im Extremfall zu einer Instabilität führen. Für den neuen Byteflight-Bus gilt Ähnliches. Er arbeitet mit einer flexiblen TDMA-Arbitrierung (FTDMA), bei der ein nicht benötigter „Slot“ zur Laufzeit drastisch verkürzt wird. Diese Flexibilität nutzt die Bandbreite besser, allerdings geht damit eine Varianz des Buszykluses einher[10]. Rückwirkungsfrei sind hingegen nur die Busse TTP/C, FlexRay und TTP/A.

Skalierbarkeit und Offenheit

Zudem unterscheidet sich TTP/A in der Skalierbarkeit und Offenheit von allen anderen Bussystemen. Die TTP/A-Architektur sieht eine Gliederung des Protokolls in drei Bausteine vor („Basic-TTP/A“, „Extended-Reliability“, „Interface-File-System“). Je nach Anwendung, kann man den einen oder anderen Baustein (bis auf „Basic-TTP/A“) weglassen, was eine Anpassung mit wenig „Overhead“ für viele unterschiedliche Anwendungen erlaubt (Kapitel 3). Außerdem besteht keine Interferenzgefahr, wenn ein Anwender anstelle des „Interface-File-System“-Bausteins z.B. ein spezielles „Gateway“ integrieren möchte. Oder den Baustein „Extended-Reliability“ mit einer „Forward-Error-Correction“ (FEC) tauscht. Dadurch ist TTP/A sehr flexibel und offen für Innovationen. Bei den Betriebssystemen wird dieses Konzept seit vielen Jahren erfolgreich angewandt (VxWorks, RTEMS, µCOS, Linux). Mit TTP/A soll ein erster Versuch in diese Richtung auch bei den Bussystemen gestartet werden.

1.3 Vorteile von Standardbauteilen

Ein weiteres Ziel des Forschungsprojekts TTSB ist die ausschließliche Verwendung von Standardbauteilen für den neuen Sensor/Aktorbus TTP/A. Im Speziellen Standard-µCs, -DSPs, -„Transceiver“, -Kabel und -Steckverbinder. Mit Standardbauteilen lassen sich neue Systeme schneller aufbauen als mit kundenspezifischen ICs (ASICs) – die Probleme werden dadurch nicht leichter, sondern anders. Insgesamt wurde für die Umsetzung und Verfeinerung des Basisprotokolls von TTP/A nur ein Mannjahr benötigt, womit sich der Ansatz rentiert hat. Außerdem sind, die auch COTS („Commercials of the Shelves“) genannten, Standardbauteile preisgünstig und hochverfügbar. Aufgrund ihres Einsatzes in zahlreichen Geräten, werden sie in großen Stückzahlen und in der Regel von mehreren Herstellern

[9] jitter (engl.) = zittern, nervös

[10] Theoretisch kann die Varianz sehr groß werden. Die Dauer eines Zykluses beträgt 250 µs. Ein „Slot“ hingegen kann 4,6..16,6 µs lang sein [79] [88]. Daraus folgt eine Anzahl von 15..54 „Slots“ pro Zyklus.

produziert. Zu diesen wesentlichen Vorteilen gesellen sich reichliche Informationen in Form von Datenblättern, Anwendungsbeispielen, Beispielprogrammen etc. Und schließlich existiert für Standardbauteile ein breiter Zubehörmarkt. Alles zusammen erleichtert den schwierigen Einstieg eines Sensor/Aktorbusses in die Kosten-Stückzahl-Spirale. Wenn dieser Punkt einmal überwunden ist, kann man über die hohen Investitionskosten für einen ASIC immer noch nachdenken. Es gibt jedoch auch Firmen, die diesen Schritt von vornherein wagen. Der neue PPC-Bus von Power-Packer und Cherry, für „low-cost“ Anwendungen im Kfz, ist ein Beispiel dafür (siehe 2.3.4). Allerdings ist dieser Weg sehr risikoreich. Erfahrungsgemäß dauert es nämlich einige Zeit und viele Iterationsschritte, bis ein Bussystem ausgereift ist und die Anwender damit zufrieden sind. Ein Software-Ansatz mit Standardbauteilen, so wie bei TTP/A, ist diesbezüglich offen.

1.4 Tendenz zur Normung

Am Ende des ersten Kapitels soll der Normungs- bzw. Standardisierungsaspekt noch kurz beleuchtet werden. Er hat für neue Bussysteme eine besondere Bedeutung. Ohne eine Norm bzw. einen Standard hat ein neues Bussystem heute kaum eine Chance sich zu etablieren. Die Anwender setzen mehr und mehr standardisierte Busse ein [16]. Defacto-Standards werden ebenfalls akzeptiert. Allerdings haben es neue Bussysteme damit schwer. Deshalb existiert für sie quasi ein „Normungszwang“. Wie Tabelle 1.1 zeigt, gibt es so gut wie keinen etablierten, ungenormten Feldbus. Und das hat seine guten Gründe, schließlich kostet eine Normung viel Zeit und Geld. Einer der Gründe liegt in der Natur des Menschen Fehler zu machen. Deshalb verlangen die Normungsgremien (CENELEC, ISO, IEC, OMG...) die Offenlegung und Prüfung eines Sachverhalts. Dadurch sinkt die Fehlerwahrscheinlichkeit, was zur Verbesserung der Qualität und Sicherheit beiträgt. Besonders auf Letzteres legen die Anwender heute mehr Wert als noch vor ein paar Jahren. Die Ursachen liegen vor allem bei der gestiegenen Produkthaftung und dem Sicherheitsbedürfnis aufgrund der intelligenten, autonomen Systeme. Nicht umsonst werden z.B. die neuen Busse BST und Planet, für Insassenschutzeinrichtungen in Kfzs, erst von der ISO geprüft und standardisiert, bevor sie zum Einsatz kommen.

Des Weiteren bieten genormte Bussysteme einen gewissen Investitionsschutz für Anwender und Hersteller. Die Einführung eines Busses, Entwicklung oder Produktion eines neuen Busses verlangt hohe Investitionen. Genormte Busse bieten in der Regel über viele Jahre eine Konstanz, wodurch die Gefahr einer Fehlinvestition, auch bei längeren Amortisationszeiten, sinkt. Denn im Gegensatz zu einem proprietären Bus, kann ein standardisierter i.A. nicht geändert werden. Bisher flossen Änderungen als abwärtskompatible Ergänzungen ein, wodurch die dann alten Normen gültig bleiben. Ein relativ aktuelles Beispiel ist die Verdopplung der „Slaves“ beim ASI-Bus; alter und neuer Standard arbeiten problemlos zusammen (siehe 2.3.1). Daneben können Normen auch zum wirtschaftlichen Wachstum beitragen. Wegen ihrer Gleichschaltungsfunktion, bauen sie Wirtschaftsschranken ab und reduzieren die Variantenvielzahl. Nicht umsonst harmonisiert die Europäische Union seit knapp zehn Jahren die unterschiedlichen Normen in Europa. Aktuelle Beispiele sind der Euro und das CE-Zeichen.

Manchmal meint man es mit den Normen aber auch zu gut. So stellt z.B. die europäische Feldbusnorm EN 50170 nur ein Sammelsurium bereits vorhandener Normen dar. Glücklicherweise gelingen solche Versuche nur selten, wie das Beispiel IEC 61158 zeigt. Über 14 Jahre lang wurde damit eine gemeinsame, internationale Feldbusnorm verfolgt. 1998 lehnten die nationalen Normungsausschüsse die IEC 61158 jedoch ab, damit laut NAMUR[11] (Interessengemeinschaft Prozessleittechnik der chemischen und pharmazeutischen Industrie) der Wettbewerb erhalten bleibt [4]. Insgesamt betrachtet über-

[11] Ehemals Normenarbeitsgemeinschaft für Mess- und Regelungstechnik in der Chemischen Industrie; Gründung am 3. Nov. 1949.

wiegen die Vorteile einer Norm. Deshalb haben die Projektpartner TTTech AG und Technische Universität Wien bei der „Object Management Group“ (OMG)[12] einen internationalen Standardisierungsvorschlag für TTP/A eingereicht [17].

Bussystem	Norm
ASI	EN 50295, IEC 62026
Profibus	EN 50170, IEC 61158, IEC 61158-2 (PA), DIN 19245-1/2, DIN 19245-3 (DP), DIN 19245-4 (PA)
CAN	EN 50325, ISO 11898
LON	IEC 62026
Ethernet	IEEE 802.3, ISO 8802/3
Interbus	DIN 19258, EN 50254, IEC 61158
Foundation Fieldbus	IEC 61158, ISA SP50
WorldFIP	EN 50170, IEC 61158
P-Net	EN 50170, IEC 61158, DK 502058, DK 502066
Firewire	IEEE 1394
Sercos	IEC 61491, EN 61491
TCN	IEC 61375
USB	Defacto-Standard
Bluetooth	möglicherweise Defacto-Standard

Tabelle 1.1: Normen relevanter Feld- und ähnlicher Busse

1.5 Ziele und Gliederung dieser Arbeit

Herrn Prof. Kopetz von der Technischen Universität Wien sind die Idee und erste Vorschläge für den neuen Sensor/Aktorbus TTP/A zu verdanken. Gegenstand dieser Arbeit ist einerseits zu zeigen, dass die Bedeutung von Sensor/Aktorbussen wächst und, welche Eigenschaften ein moderner Sensor/Aktorbus haben sollte. Dazu wurden in den vorangegangenen Abschnitten die Vorteile von Sensor/Aktorbussen, aktuelle Probleme und Neuentwicklungen diskutiert.

Anderseits sollen Trends und die Schwächen existierender und zukünftiger Sensor/Aktor- oder vergleichbarer Busse dargelegt werden, um die Bedeutung von TTP/A zu unterstreichen. Das geschah zum Teil in *Kapitel 1* und wird anhand von detaillierten Beispielen in *Kapitel 2* vertieft, insbesondere die Eigenschaften (Real-)Zeitverhalten, Effizienz und Verlässlichkeit. Am Ende von *Kapitel 4* befindet sich eine Tabelle, die einen übersichtlichen Vergleich zwischen TTP/A und vergleichbaren Bussystemen gestattet.

Weitere Ziele dieser Arbeit sind konzeptionelle Beiträge zum TTP/A-Protokoll und seiner Architektur zu leisten, damit der neue Sensor/Aktorbus absolut zeitdeterministisch ist, auf Standardbauteilen basiert (insbesondere µCs mit Hardware-UART), effizient arbeitet, für einen Anwender skalierbar und offen ist, eine angemessene Fehlersicherung besitzt und der ereignisorientierte „Master/Slave“-Anteil stärker in den Vordergrund rückt. Auf diese Punkte geht *Kapitel 3* ein.

[12] Die OMG wurde 1989 ins Leben gerufen, mit der Aufgabe einen allgemeinen Architekturrahmen für objektorientierte Anwendungen mit einer breiten Schnittstellenspezifikation zu schaffen. Heute zählt die OMG mehr als 800 Mitglieder.

Zu den wichtigsten Zielen dieser Arbeit zählt der Entwurf eines Realisierungskonzepts für das Basisprotokoll von TTP/A („Basic-TTP/A“) mit den Randbedingungen: nur Software, keine Spezialbauteile, moderate Rechenleistung, wenig Speicher, wenige Pins und möglichst hohe Baudraten. Es wird in *Kapitel 4* vorgestellt, wobei den hohen Realzeitanforderungen des Protokolls, einem UART-Jitter, einer impliziten Synchronisation, einem „Interleave“-Algorithmus, einer speziellen „Task“-Aufteilung und einer Selbstjustierung das Hauptaugenmerk gilt.

In *Kapitel 5* wird anhand von C16x-µCs beispielhaft gezeigt, dass sich die erarbeiteten Lösungsstrategien umsetzen lassen. Es wird die erste funktionierende und portable C-Implementierung für „Basic-TTP/A“ vorgestellt. Ein Demonetz beweist, dass man mit „Basic-TTP/A“ sowohl einfache als auch anspruchsvolle Anwendungen bedienen kann. Ferner liefern die praktischen Erfahrungen Zahlenwerte über den tatsächlichen Speicherbedarf und die baudratenbezogene Rechenleistung.

Da „Basic-TTP/A“ nicht für verlässliche Systeme geeignet ist aber die Voraussetzungen dafür erfüllt, ist die Erarbeitung eines Redundanzkonzepts unter Berücksichtigung der Randbedingungen das letzte Ziel dieser Arbeit. *Kapitel 6* stellt diesbezüglich spezielle fehlersichernde Codes und abgestufte topologische Erweiterungen für das Basisprotokoll vor.

Ein Resümee dieser Forschungsarbeit und einen Ausblick findet man in *Kapitel 7*. Es werden die wichtigsten Ergebnisse aufgezählt und kurz auf das geplante Forschungsvorhaben „Smart-Transducers“ eingegangen. Eine Teilaufgabe dieses Forschungsvorhabens ist die Umsetzung und Erprobung des Redundanzkonzepts. Außerdem wird die Idee einer neuen TDMA-Variante, das so genannte „Carrier-Sense“-TDMA, vorgestellt.

Den Abschluss bilden der Anhang in *Kapitel 8* und das Literaturverzeichnis in *Kapitel 9*. Dort befinden sich nützliche Informationen über Oszillatoren, µCs, Normen, Schaltpläne, eine CD mit den TTP/A-Quellcodes, weiteren Informationen, Bildern und Videos und weiterführende Literaturstellen.

2 Stand der Technik: Sensor/Aktorbusse

2.1 Eigenschaften der Sensor/Aktorbusse

Einige Eigenschaften der Sensor/Aktorbusse wurden bereits in Kapitel 1 behandelt. Der Vollständigkeit halber werden sie hier noch einmal erwähnt, aber nicht mehr erklärt. Sensor/Aktorbusse sind kleine, einfache Feldbusse für den preisgünstigen Anschluss von Sensoren, Aktoren und Einzelsignalen an einen technischen Prozess. In der Vernetzungshierarchie der Automatisierungstechnik sind sie ganz unten in der Feldebene angesiedelt (Bild 2.1). Dort ergänzen sie die Feldbusse für eine durchgängige Vernetzung bis in die Sensoren und Aktoren hinein. Außerdem entlasten Sensor/Aktorbusse die Feldbusse von lokalen Nachrichten und Daten. In anderen Sparten liegen die Sachverhalte ähnlich.

Eine scharfe Abgrenzung zwischen Sensor/Aktor- und Feldbussen gibt es nicht. Genauso wenig wie zwischen den anderen Bussystemen, da Herstellerfirmen und Nutzerorganisationen ständig versuchen die Anwendungsdomänen anderer Busse zu erobern. Das in den 70er Jahren von der Firma Xerox entwickelte Ethernet ist das beste Beispiel: Mit dem Einzug in die PCs hat es die Betriebsebene vollständig erobert und durch den Preisverfall Stück für Stück die Leitebene. Heute ist es so günstig und schnell geworden, dass es auch zunehmend in der Feldebene eingesetzt wird. Für den Sensor/Aktorbereich ist es allerdings nach wie vor zu teuer und zu aufwändig. Niemand würde eine Handvoll intelligente Sensoren und Aktoren über einen „Switch“ sternförmig vernetzen. Ein „Switch“ ist dafür schlichtweg zu teuer; abgesehen von den Kabelkosten und den armdicken Kabelbäumen.

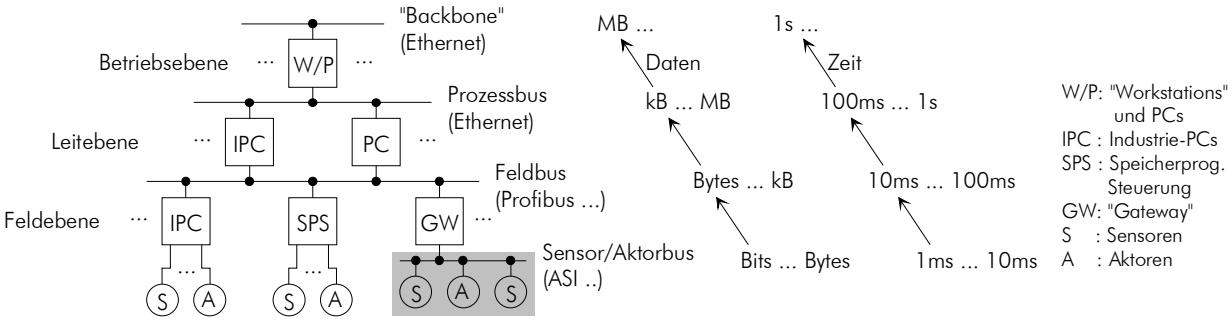


Bild 2.1: Vernetzungshierarchie in der Automatisierungstechnik

Wie Bild 2.1 zeigt, sind die Datenmengen in der Sensor/Aktorebene am kleinsten. Sie liegen im Bit- und unteren Bytebereich. Bits werden am häufigsten übertragen, weil das Gros der Sensoren und Aktoren binär ist. Ein bis zwei Bytes braucht man hingegen für analoge Sensoren und Aktoren, Diagnose- und Konfigurationsdaten[13]. Die Anforderungen an das Zeitverhalten sind in der Sensor/Aktorebene am größten. Viele technische Prozesse besitzen Zeitkonstanten zwischen 10..100 ms. Damit die Totzeit durch einen Sensor/Aktorbus für eine Steuerung oder Regelung keine Rolle spielt, müssen die Daten wesentlich schneller übertragen werden (1..10 ms). Aus Kosten- und EMV-Gründen, kann man die Baudrate bei einem Sensor/Aktorbus aber nicht beliebig hoch machen (< 500 kBit/s). Deswegen sollte die Übertragungseffizienz möglichst groß und folglich die Telegramme und Pausen möglichst klein sein – leider trifft das nicht für alle Sensor/Aktorbusse zu. Man erreicht das durch eine Arbitrierung mit wenig „Overhead“ und einer angepassten Fehlersicherung.

Bezüglich des Arbitrierungs-„Overheads“ ist das TDMA-Verfahren gut geeignet. Es ist einfach, kostengünstig umsetzbar und sehr effizient. Da es jedoch unflexibel ist, wird es kaum angewandt (Sercos, TTP/C). Das „Master/Slave“-Verfahren ist ebenfalls einfach und kostengünstig realisierbar. Im Gegensatz zum TDMA-Verfahren ist es sehr flexibel, aber nicht besonders effizient. Da die Flexibilität mehr wiegt als die Effizienz, wird es sehr häufig eingesetzt. Fast alle Sensor/Aktorbusse arbeiten nach dem „Master/Slave“-Verfahren (ASI, LIN, BST, PPC, DIN-Bus, ISO 9141, DSI[14]...). Der TTP/A-Bus dagegen nutzt die Vorteile beider Verfahren, indem er eine Mischform verwendet. Andere Arbitrierungsverfahren, wie z.B. CSMA (CAN) oder FTDMA (Byteflight), sind für Sensor/Aktorbusse zu aufwändig und zu teuer.

Für die Fehlersicherung ist in erster Linie die Restfehlerwahrscheinlichkeit ausschlaggebend. Sie ist umso niedriger, je größer die Hamming-Distanz und je kürzer die Telegrammlänge ist. Außerdem spielt der Datenmodus eine Rolle. Periodisch aktualisierte Daten, so genannte idempotente Daten, sind robuster als einmalige Daten, weil Übertragungsfehler nur temporär wirken. Durch die Kombination aus Idempotenz und kleinen Datenpaketen, kann man mit wenigen Sicherungsbits eine vernünftige Fehlersicherung erreichen. Oftmals genügt schon ein Paritybit und ein periodischer Busbetrieb, um die Kriterien der ersten Datenintegritätsklasse (I1) nach DIN 19244 zu erfüllen. Genau aus diesem Grund arbeiten die meisten Sensor/Aktorbusse für einfache Anwendungen auf diese Art und Weise. Sicherheitskritische Aufgaben verlangen allerdings mehr. Wegen der, aus Kostengründen, oft eingesetzten drahtgebundenen Übertragung, eignet sich eine CRC-Sicherung mit ca. 4..8 Prüfbits (Hamming-Distanz ca. 3..6).

Die Netzausdehnung von Sensor/Aktorbussen liegt zwischen 10..100 m. Mehr wird i.A. nicht benötigt, weil Sensoren und Aktoren typischerweise zu lokalen, anwendungsorientierten Gruppen zusammengefasst werden (Schaltschrank, Autositze, Wohnung...). Aus diesem Grund übersteigt die Teilnehmerzahl pro „Sub“-Netz selten einen Wert von 30..40. Dementsprechend einfach und günstig ist auch die Topologie. Sensor/Aktorbusse weisen durchgehend eine Linienstruktur auf – der ringförmige Interbus-S wird hier zu den Feldbussen gerechnet. Wegen dieses einfachen Aufbaus, besitzen Sensor/Aktorbusse nur einen „Physical-Layer“ (OSI-Schicht 1), „Data-Link-Layer“ (OSI-Schicht 2) und manchmal auch einen „Application Layer“ (OSI-Schicht 7)[15].

[13] Das Verhältnis von Sensoren zu Aktoren beträgt in etwa 2:1 [43].

[14] Eine „Full-Duplex“-Kommunikation macht den DSI-Bus etwas effizienter.

[15] Bei den Feldbussen ist das nicht anders. Lediglich der PNET- und LON-Bus besitzen einen „Network-Layer“ (OSI-Schicht 3) für große, komplexe Busstrukturen.

2.2 Allgemeine Busübersicht und Trends

2.2.1 Automatisierungstechnik

Sensor/Aktorbusse

In der Automatisierungstechnik ist der ASI-Bus der einzige etablierte, standardisierte Sensor/Aktorbus. Manchmal werden auch die Bussysteme Profibus-DP und Interbus-S für die Vernetzung von Sensoren und Aktoren verwendet; der Bitbus von Intel und die SLIO-CAN-Variante sind obsolet. Als allgemeine Feldbusse gestatten sie den Anschluss von Sensoren und Aktoren über Sammelmodule. Mit einer direkten Vernetzung oder mit intelligenten Sensoren und Aktoren hat das jedoch wenig zu tun. Deshalb werden sie in dieser Arbeit zur Gruppe der Feldbusse gezählt. Neben dem ASI-Bus existieren ein paar proprietäre bzw. firmenspezifische Sensor/Aktorbusse, wie z.B. der OPUS-Bus von der Firma Marquardt oder der Sensoplex-Bus von der Firma Truck [12]. Diese Busse haben keine Zukunft, weil die Anwender zu standardisierten Bussystemen tendieren (siehe 1.4). Neuerungen könnten sich, so wie einst beim CAN-Bus, durch die Innovationen auf dem Kfz-Sektor ergeben. Kandidaten sind z.B. der LIN- und TTP/A-Bus. Letzterer wurde z.B. auf der Automatisierungsmesse „Interkama 2001“ in Düsseldorf, auf dem Stand des Projektpartners Universität Stuttgart, vorgestellt. Dort hat er bei einigen Firmen reges Interesse geweckt.

Feldbusse

Bei den Feldbussen dominiert der Profibus, der Interbus-S und der CAN-Bus. Noch, denn seit ein paar Jahren haben diese Bussysteme eine ernsthafte Konkurrenz durch das Ethernet bekommen. Mit dem Siegeszug der PCs wurde es zum weltweiten Standard-LAN. Dadurch ist es sehr günstig geworden, obwohl es technisch aufwändig ist. Eine einzelne Ethernet-Karte kostet ca. zwischen 20 und 40 €, eine Feldbuskarte dagegen viele 100 €. Dieser Unterschied wird noch größer, wenn man einen PC mit integriertem Ethernet verwendet. Bei den, für die Feldebene konstruierten, IPCs ist das standardmäßig der Fall. Bezüglich der Geschwindigkeit lässt das Ethernet keine Wünsche offen. Die heute gängige 100 MBit/s-Variante würde vollends ausreichen. Sie wird jedoch schon vom 1 GBit/s-Ethernet abgelöst und das 10 GBit/s-Ethernet befindet sich gerade in der Spezifikation. Viel wichtiger als die Geschwindigkeit ist die Realzeitfähigkeit. Hier hatte das Ethernet, wegen den typischen Buskollisionen durch die CSMA/CD-Arbitrierung, Probleme, bis zur Einführung der „Switch“-Technologie. Die „Switches“ lösen das Realzeitproblem, indem sie die Telegramme intern über ein Zeitscheibenverfahren ordnen. Gekoppelt mit einer Sterntopologie bauen sie so Vollduplex-Punkt-zu-Punkt-Verbindungen zwischen den Teilnehmern auf. Vorausgesetzt der interne „Switch“-Bus arbeitet schnell genug, was in der Regel zutrifft, so ist das Realzeitproblem des Ethernets gelöst. Auf dieser Technik basiert das so genannte „Industrial-Ethernet“ und das JetWeb der Firma Jetter [94]. Der Unterschied zwischen beiden ist im Wesentlichen der Internet-Ansatz bei JetWeb. Er ist jedoch marginal, falls überhaupt vorhanden, da dieser Weg durch die Globalisierung sowieso vorgezeichnet ist.

Sicherheitsbusse

Aufgrund des gestiegenen Sicherheitsbedürfnisses (siehe Kapitel 1), wurden für die meisten Feldbusse Sicherheitsvarianten entwickelt. So entstanden in den letzten Jahren das ProfiSafe-Profil, der Interbus-Safety, das „Safety-at-Work“-Konzept (ASI-Bus), der SafetyBus (CAN-Bus), und das DeviceNet-Safety, um die Wichtigsten zu nennen. Viele der Sicherheitsvarianten haben bereits eine TÜV-, BIA[16]-

[16] Berufsgenossenschaftliches Institut für Arbeitssicherheit

oder anderweitige Zertifizierung. Das Ethernet ist als Sicherheitsbus jedoch nicht geeignet. Wegen der CSMA/CD-Arbitrierung, können EMV-Störungen zu Telegrammwiederholungen führen und somit „Deadlines“ verletzen (siehe Abschnitt 1.2). Voraussichtlich wird sich das in der Zukunft ändern. Durch die „Switch“-Technik ist die Arbitrierung im Ethernet überflüssig geworden, weil bei einer Vollduplex-Punkt-zu-Punkt-Verbindung keine Kollision auftreten kann. Die ersten Netzwerkkarten mit abschaltbarer CSMA/CD-Arbitrierung existieren bereits.

2.2.2 Kraftfahrzeugtechnik

Einfache Sensor/Aktorbusse

Innovationsfreudiger als die Automatisierungstechnik ist die Kfz-Technik. Sie hat in den vergangenen Jahren einige neue Bussysteme hervorgebracht bzw. Projekte dafür gestartet. Im Sensor/Aktorbereich werden diese Busse oftmals als NANs („Niche Area Network“), Sensorbusse, Zündbusse oder „Subnets“ bezeichnet. Einfache Ausführungen werden für die Komfort- (Sitzheizung, Fensterheber...) und Karosserieelektronik (Lampen, Scheibenwischer...) benötigt. Bei diesen Sensor/Aktorbussen spielt in erster Linie der Preis eine Rolle. Gemäß [71] darf er pro Teilnehmer nicht über 0,5..1 € liegen. Denn der Kunde ist nicht willig einen Euro mehr für einen Sensor/Aktorbus auszugeben, solange die nahezu gleiche Funktionalität in konventioneller Technik von der Konkurrenz günstiger angeboten wird. Die technischen Vorteile in der Produktion oder beim Service durch „Plug&Play“, die höhere Zuverlässigkeit im Betrieb wegen der reduzierten Kabelbäume und Kontakte und geringere Treibstoffkosten aufgrund eines intelligenten Energiemanagements interessieren nur wenige Kunden. An den Sensor/Aktorbussen führt mittelfristig jedoch kein Weg vorbei, auch wenn der Preisdruck momentan sehr hoch ist. In konventioneller Technik steigen die Gesamtkosten durch die raschen Neuerungen und steigenden Anforderungen rapide an. Heute basieren im Mittel 28 von 33 Innovationen in Kfzs der Oberklasse auf Elektronik. Schätzungen nach werden die Elektronikkosten in Autos bis 2005 von derzeit 20..26 % auf über 30 % ansteigen [92]. Die S-Klasse von Mercedes z.B. besitzt schon jetzt 35 Steuergeräte [143]. Aus diesem Grund bedarf es einer strukturellen Änderung durch intelligente, vernetzte Sensoren und Aktoren. Neben vielen Vorteilen, tragen sie vor allem mittelfristig zur Kostenreduktion durch eine einfachere und schnellere Produktion, Wartung und Entwicklung bei.

An einfachen Kfz-Bussystemen existiert der firmenspezifischen A-Bus (VW), M-/I-/P-Bus (BMW), Dotnet-Bus (Dotronic), UART-Bus (General Motors), CCD-Bus (Chrysler), ACP-Bus (Ford) und einige mehr [12] [71]. Genormt sind hingegen der ISO 9141-Bus und die amerikanischen J-Busse (J1850, J1708, J1587...)[17]. Die Einsatzgebiete dieser Bussysteme sind sehr speziell. Meistens dienen sie als Diagnose- und/oder Konfigurationsschnittstelle für Steuergeräte. Hie und da werden sie zur Vernetzung von Cockpit-Instrumenten (z.B. BMW) oder ähnlichen Dinge eingesetzt. Für moderne Sensor/Aktoranwendungen sind sie ungeeignet. Teils sind sie zu speziell, teils zu teuer, teils zu alt (über 15 Jahre) und vor allem sind die meisten proprietär. Ansonsten würden die Automobilhersteller, Zulieferer und „Chip“-Fabrikanten kein Geld für die Entwicklung neuer, kostengünstiger, flexibler, genormter Sensor/Aktorbusse ausgeben. Derzeit entstehen der LIN-Bus (siehe Abschnitt 2.3.3), PPC-Bus (siehe Abschnitt 2.3.4) und DC-Bus für einfache Sensor/Aktoranwendungen im Kfz. Der TTP/A-Bus ist zwar keine kommerzielle Entwicklung, aber er eignet sich hierfür ebenso.

Am weitesten fortgeschritten ist der, von den Firmen Audi, BMW, DaimlerChrysler, Volvo, VW, VCT und Motorola entwickelte, LIN-Bus. Er arbeitet nach dem „Master/Slave“-Prinzip und verwendet den ISO 9141-„Physical-Layer“ mit Baudraten bis zu 20 kBit/s. Erste LIN-Bus-ICs sind bereits auf

[17] Manche sind in Europa nicht zugelassen.

dem Markt erhältlich. In direkter Konkurrenz dazu steht der PPC-Bus von den Firmen Power-Packer (Niederlande) und Cherry (weltweit bekannt durch seine Tastaturen). Auch er ist ein einfacher „Master/Slave"-Bus mit ISO 9141-„Physical-Layer" und Baudraten bis 32 kBit/s. Der DC-Bus von Yamar (Israel) ringt ebenso um diese Anwendungsdomäne. Seine Besonderheit ist der Datentransport über die Kfz-Stromleitungen. Dabei unterscheidet der DC-Bus zwischen mechatronischen, telematischen und Multimedia-Anwendungen. Für den hier behandelten „low-cost" Bereich ist die mechatronische Variante bestimmt. Sie arbeitet mit einer ASK-Modulation und einer festen Geschwindigkeit von 10 kBit/s [56]. Telematische Anwendungen werden dagegen DQPSK-moduliert und mit 250 kBit/s betrieben [57]. Als Arbitrierungsverfahren setzt der DC-Bus eine CSMA-Methode ein (CSMA/CR[18]). Der Verbindungstypus ist durchwegs „Peer-to-Peer", da beim DC-Bus alle Teilnehmer gleichwertig sind. Über den Multimedia-Bereich ist momentan nur die geplante Baudrate von 1,7 MBit/s bekannt [54].

Sicherheits-Sensor/Aktorbusse

Diese Gruppe von Sensor/Aktorbussen ist neu. Als Domäne sind zwar alle sicherheitskritischen und verlässlichen Anwendungen im Kfz geplant. Der Focus liegt jedoch auf Unfallschutzeinrichtungen und Rückhaltesystemen, wie „Airbags", Gurtstraffer, Überrollbügel und „Crash"-Sensoren. Auch hier ist der Kostendruck mit 1..2 € pro Teilnehmer groß [71], weil im Normalfall niemand davon Gebrauch machen will. In der Entwicklung befinden bzw. befanden sich die Busse BST, Planet, SafetyBus[19], DSI, SURFS und BoTe. Der TTP/A-Bus ist für diese Anwendungen ebenfalls geeignet, bleibt als Forschungsprojekt jedoch außen vor. Ein gängiger Ausdruck für diese Sicherheits-Sensor/Aktorbusse ist Zündbus. Er stammt von den pyrotechnischen Aktoren, die bei einem Unfall gezündet werden. Anders kann man die in kurzer Zeit benötigte Energie zum Aufblasen von „Airbags", Straffen von Gurte etc. bei kleinem Volumen momentan nicht aufbringen[20].

Der DSI-Bus („Distributed Systems Interface") ist eine Entwicklung der Firmen Motorola und TRW (USA). Er kam bereits zum Einsatz, wurde aber nach kurzer Zeit wieder zurückgezogen; wahrscheinlich wird er aus Sicherheitsgründen überarbeitet (siehe Punkt 1.2). Die Kommunikation findet beim DSI-Bus nach „Master/Slave"-Manier statt. Interessant ist eine Vollduplex-Verbindung zwischen „Master" und „Slave". Dazu versendet der „Master" seine Telegramme spannungsmoduliert, währenddessen die „Slaves" strommoduliert antworten. Natürlich geht das nicht gleichzeitig, weil die „Slaves" erst nach dem „Master"-Aufruf wissen, wer angesprochen ist. Aus diesem Grund erfolgt die „Slave"-Antwort um ein Telegramm versetzt, d.h. im nächsten „Master"-Aufruf. Mit dieser Methode nutzt der DSI-Bus seine Bandbreite besser als die anderen Bussysteme in dieser Domäne. Für den Gleichlauf der Teilnehmer sorgt eine „Master"-taktsynchrone Übertragung[21], die der Vollduplex-Betrieb ermöglicht. Das Busmedium ist eine Zweidrahtleitung, mit dem Daten und Energie gleichzeitig transportiert werden. Um eine richtige „Powerline" handelt es sich hierbei nicht, weil der „Master" die Energiequelle ist. Für die Datenübertragung senkt er die Betriebsspannung auf verschiedene Pegel ab; die „Slaves" machen das analog mit ihrer Stromaufnahme. Dadurch wird seine Endstufe stark belastet und die mittlere Stromabgabe auf ca. 100 mA beschränkt. Als minimale Baudrate wurden 5 kBit/s spezifiziert. Nach oben hin steckt die Spezifikation ausdrücklich keine Grenze ab [64]. Aufgrund der vermutlichen Überarbeitung, wird sich die für 2002 geplante Einführung laut [65] wohl etwas verschieben.

---

[18] „Carrier Sense Multiple Access/Collision Resolve"

[19] Nicht zu verwechseln mit dem SafetyBus der Firma Pilz.

[20] Bei einem Frontalaufprall muss eine Zündung innerhalb von 50 ms erfolgen, bei einem Seitenaufprall innerhalb von 3 ms; der „Airbag" selbst expandiert in 30 ms [23].

[21] Manchester-Code mit einem Pegelwechsel nach 1/3 oder 2/3 Bitdauer.

Eine Mischung aus den beiden Zündbussen BoTe (Bosch, Temic) und SURFS (Siemens) ist der BST-Bus (Bosch, Siemens, Temic). Er funktioniert ähnlich wie der DSI-Bus, weil er ebenfalls mit der „Master/Slave"-Methode arbeitet und einen differentiellen Spannungshin- und Stromrückkanal verwendet. Außerdem ist das Prinzip der Kommunikation und Energieübertragung nahezu gleich. Die Möglichkeit der Vollduplex-Kommunikation nutzt er jedoch nicht. Anstelle von Daten verschickt der „Master" Leertelegramme, wenn ein „Slave" sendet [67]. Seine max. Baudrate ist mit 250 kBit/s spezifiziert. Momentan wird der BST-Bus durch die ISO genormt. Weitere Details befinden sich unter Punkt 2.3.5.

Mit dem Planet-Bus versucht die Firma Philips in diesem Zukunftsmarkt präsent zu werden. Nach seiner Überarbeitung (siehe Abschnitt 1.2) weist der Planet-Bus einige Parallelen zum BST-Bus auf. Der Planet-Bus verwendet eine Zweidrahtleitung für die gleichzeitige Übertragung von Daten und Energie. In einer so genannten „Power Phase" ($U_{diff} > 10$ V) beziehen die „Slaves" Energie vom Bus. Für „Master"-Telegramme ist die „Data-Phase 1", mit negativen Spannungspegeln ($U_{diff} < 0$ V), reserviert. „Slaves" hingegen antworten in der „Data-Phase 2". Dazu überträgt der „Master" seinen Bittakt bei einem Spannungspegel $U_{diff} \approx 5$ V. Ein sendender „Slave" moduliert diese Spannung zwischen 1,5 und 5 V im Takt des „Masters". Sollte kein „Slave" senden, wird dieser Zustand in der Spezifikation „Clock Phase" genannt. Die Baudrate des Planet-Busses kann wie beim BST-Bus bis zu 250 kBit/s betragen. Eine ISO-Normung ist ebenso vorgesehen [66] [68].

Der vorläufig letzte Zündbus ist der SafetyBus von Delphi Automotive Systems (kurz Delphi). Leider ist die Bezeichnung etwas unglücklich gewählt, weil es bereits einen SafetyBus von der Firma Pilz für die Automatisierungstechnik gibt. Der Delphi-SafetyBus befindet sich in der Entwicklung, weshalb wenige Informationen öffentlich sind [71]. Bekannt ist, dass er nach dem „Master/Slave"-Prinzip arbeitet, die Baudrate bis 500 kBit/s reicht, und eine RTZ-Bitcodierung verwendet wird. Da diese Daten mit dem ehemaligen Zündbusversuch „Delphi Custom Protocol" übereinstimmen, könnte der Delphi-SafetyBus der Nachfolger sein.

„X-by-Wire"-Busse

Eine völlig andere Klasse bilden die „X-by-Wire"-Busse für Kfzs. Sie sind für sicherheitskritische und verlässliche Anwendungen auf Geräteebene, wie „Steer-by-Wire" und „Break-by-Wire", konzipiert. Ihre Ansprüche bezüglich Sicherheit, Verlässlichkeit, Fehlertoleranz und Bandbreite liegen deutlich über denen der Sensor/Aktorbusse. Aus diesem Grund dürfen die „X-by-Wire"-Busse entsprechend teurer sein. Die Kosten pro Knoten[22] werden auf ca. 15 € geschätzt [71]. Für die „X-by-Wire"-Anwendungen wurden die neuen Bussysteme TTP/C von TTTech und Byteflight von BMW geschaffen. In der Entwicklung befindet sich der FlexRay-Bus des Firmenkonglomerats: BMW, DaimlerChrysler, Motorola, Philips, Bosch und General Motors. Die drei Bussysteme visieren zwar die wirtschaftlich interessante Kfz-Technik an, sie sind jedoch auch für sicherheitskritische Anwendungen in Flugzeugen, Zügen oder Industrieanlagen geeignet.

Zur Zeit wird der Byteflight-Bus als Einziger serienmäßig in Kfzs eingesetzt. Er verbindet 13 passive Sicherheitssysteme des so genannten „Intelligent Safety Integration Systems" (ISIS) im neuen 7er BMW [79]. Der Byteflight-Bus arbeitet mit einem optischen „Physical-Layer" in Sterntopologie. Die Baudrate beträgt 10 MBits/s. Als Arbitrierungskonzept verwendet er FTDMA („Flexible Time Division Multiple Access"). FTDMA überträgt Daten, so wie TDMA, in „Slots". Diese sind jedoch nicht fest, sondern werden drastisch verkürzt, wenn keine Daten vorliegen. Dadurch kann FTDMA die zur Verfügung stehende Bandbreite in der Regel besser nutzen als TDMA. Des Weiteren arbeitet der Byteflight-Bus nachrichtenorientiert. In einem „Slot" befindet sich ein Telegramm mit einem „Identi-

[22] Der Ausdruck Knoten (engl. node) ist die fachübliche Bezeichnung für einen Teilnehmer.

fier“ und bis zu zwölf Datenbytes [88]. Anhand des „Identifiers“ entscheidet ein Teilnehmer selbst, so wie beim CAN-Bus, ob das Telegramm bzw. die Nachricht für ihn bestimmt ist.

Mit dem TDMA-Prinzip arbeitet dagegen der TTP/C-Bus. Die Zuordnung der „Slots“ zu den Teilnehmern wird a priori über einen „Schedule“ festgelegt. Ein „Identifier“ o.ä. wird nicht benötigt. Aus diesem Grund ist die „Slot“-Effizienz hier höher als beim Byteflight-Bus. Für die Gesamteffizienz gilt das i.A. nicht. Anfänglich verwendete der TTP/C-Bus einen CAN-„Physical-Layer“ bei einer Übertragungsgeschwindigkeit von 2 MBit/s. Mit dem neuen „Controller“ AS8202 der Firma Austria Mikro Systeme hat sich das vor kurzem jedoch geändert. Jetzt beträgt die max. Baudrate 25 MBit/s [87]. Die Daten werden beim TTP/C-Bus standardmäßig über zwei parallele, linienförmige Busse redundant übertragen; der Byteflight-Bus muss das in der Applikation realisieren [91]. Außerdem bietet der TTP/C-Bus die Möglichkeit Teilnehmer redundant auszulegen („Fault Tolerant Units“) [82]. Mittels so genannter Schattenknoten kann sogar eine netzglobale Redundanz geschaffen werden. Besonders erwähnenswert ist das „Membership-Protocol“. Es übermittelt jedem der max. 48 Knoten den Zustand der anderen Knoten. Dadurch weiß jeder Knoten zu jeder Zeit, welche Knoten aktiv sind. Des Weiteren quittiert das „Membership-Protocol“ implizit Nachrichten [10], wodurch der TTP/C-Bus in Summe eine sichere und verlässliche Datenübertragung garantiert. Obwohl er noch nicht in Serie eingesetzt wird, hat er als Einziger bereits seine Befähigung für „X-by-Wire“-Anwendung in Kfz-Prototypen bewiesen. Voraussichtlich wird die Firma Audi der erste serienmäßige Anwender des TTP/C-Busses werden.

Kurioserweise ist jedoch der noch nicht existierende FlexRay-Bus der Favorit. Den bisherigen Informationen nach kombiniert er die Vorteile des Byteflight- und TTP/C-Busses [90] [80] [89]. Der FlexRay-Bus arbeitet mit einem statischen TDMA- und einem dynamischen FTDMA-Teil. Sicherheitskritische Daten werden im TDMA-Teil übertragen, weil dort harte Realzeitbedingungen herrschen. Der dynamischen FTDMA-Teil hingegen gehört den sonstigen Nachrichten und Daten (z.B. Diagnose). So kann der FlexRay-Bus die Sicherheitsvorteile von TDMA nutzen ohne starr zu sein. Zudem überträgt der FlexRay-Bus sicherheitskritische Daten über zwei parallele Kanäle redundant. Alle anderen Informationen werden dagegen einfach, aber mit der doppelten Bandbreite übertragen. Die Baudrate ist einstellbar und kann bis 10 MBit/s reichen. Es können max. 64 Teilnehmer an den Bus angeschlossen werden. Als Topologie verwendet der FlexRay-Bus eine Linie für Cu-Medien und einen Stern für optische Medien. Erste „Chips“ werden nicht vor 2004 erwartet.

Multimedia-Busse

Bussysteme dieser Gruppe haben hohe bis sehr hohe Bandbreiten. Sie müssen gut funktionieren, aber weder verlässlich, noch sicher oder hochverfügbar sein. Aus EMV-Gründen verwendet man für diese Busse optische „Physical-Layer“, weshalb die Topologie entweder ein Ring oder Stern ist. Die Kosten für einen Knoten betragen etwa 10 € [71]. Für den Kfz-Multimedia-Bereich wurden die Bussysteme D2B (Digital Data Bus) von der Firma C&C electronics und MOST (Media Oriented Systems Transport) von der Firma OASIS/SiliconSystems entwickelt. Der bereits etwas ältere D2B-Bus arbeitet mit einer Baudrate von 12 MBit/s. Sein jüngerer Konkurrent, der MOST-Bus, ist für 50 MBit/s ausgelegt. Wegen der höheren Bandbreite wird er momentan von der Automobilindustrie favorisiert [76]. Da jedoch der Bandbreitenbedarf im Multimedia-Sektor rasch wächst, ist hie und da bereits der allgemeine, 400 MBit/s schnelle Multimedia-Bus Firewire (IEEE 1394) im Gespräch. Denkbar ist auch, dass der für PCs neu entwickelte USB-Bus (Universal Serial Bus) hier Verwendung findet. Er ist mittlerweile in jedem PC, „Notebook“, Drucker, „Scanner“ oder in jeder „WebCam“ enthalten, wodurch sein Preis fällt und seine Attraktivität steigt. Die anfänglich moderate Geschwindigkeit von 12 MBit/s ist bereits auf 480 MBit/s angehoben worden.

Zwischenbusse

Aufgrund der Busvielfalt hat die SAE („Society of Automotive Engineers") Überlegungen zur Entkopplung nicht fahrzeuggebundener Geräte (Handy, Autotelefon, CD-Wechsler, Alarmanlage, Navigationssystem...) von fahrzeugspezifischen Bussystemen (CAN, J1850, D2B, MOST...) angestellt. So entstand ein Konzept mit dem Namen IDB (Intelligent Transportation System Data Bus)[23], das über einen Zwischenbus eine Standardschnittstelle schaffen soll. Dazu müssen die fahrzeugunabhängigen Geräte eine IDB-Schnittstelle besitzen und die Kfz-Busse ein IDB-„Gateway". Erste erfolgreiche Versuche in prototypisch ausgerüsteten Kfzs wurden bereits durchgeführt [77]. Seit Ende 2000 existiert ein vorläufiger SAE-Standard unter der Bezeichnung „SAE J 2366 (Draft)" [73] [74].

2.2.3 Sonstige Sparten

Neuerungen sind auch bei der Deutsche Bundesbahn zu verzeichnen. Für die Automatisierung der Züge hat die Deutsche Bundesbahn TCN („Train Communication Network") entwickelt [96]. TCN besteht aus zwei Bussystemen: MVB („Multi Vehicle Bus") und WTB („Wire Train Bus"). Der MVB-Bus befindet sich in den Wagons und Lokomotiven, währenddessen der WTB-Bus sie untereinander verbindet. Die Übertragung findet differentiell (z.B. RS485) bei Baudraten zwischen 1..1,5 MBit/s statt. Einsatzgebiete sind unter anderem sicherheitsrelevante Systeme. Mit der IEC 61375 wurde TCN gegen Ende des Jahres 1999 genormt [95].

In der Gebäudetechnik wird dagegen nach wie vor der LON- (Local Operating Network) und EIB-Bus („European Installation Bus") eingesetzt. Beide sind in existierenden Gebäuden jedoch schlecht nachrüstbar, weil Kabel verlegt werden müssen. Aus diesem Grund wird sich hier auf Dauer die wahrscheinlich kostengünstigere „Powerline"- und Funktechnik (z.B. Bluetooth) durchsetzen.

2.3 Bewertung vergleichbarer Bussysteme

2.3.1 ASI-Bus

Allgemeines

Das Aktor-Sensor-Interface (ASI-Bus) ist ein Bussystem für binäre Sensoren und Aktoren in der industriellen Automatisierungstechnik. Seine Entwicklung fand zwischen 1990 und 1993 durch elf Firmen und zwei Hochschulinstitute statt [43]. Der ASI-Bus ist ein „Master/Slave"-Bus mit moderaten Kosten. Ein „Master-Chip" kostet ungefähr 10 € und ein „Slave-Chip" ungefähr 5 €. Aus diesem Grund rentiert sich der ASI-Bus nur in größeren Betrieben, die durch den Zeitvorteil bei der Wartung und Reparatur einen relativ schnellen „Return of Investment" haben. Mit der Integration von „Master" und „Slave" auf einem „Chip" (A^2SI), will die Firma AMI Semiconductor die Preise senken. Der ASI-Bus hat eine Baumtopologie mit einer max. Ausdehnung von 100 m [24]. Seine Datenpakete sind auf binäre Sensoren und Aktoren angepasst und dementsprechend klein. Eine Besonderheit des ASI-Busses ist die eigens entwickelte APM-Technik (Alternierende-Puls-Modulation) zur Übertragung von Daten und Energie auf einem Leitungspaar. Seine ursprüngliche Spezifikation ist außerdem in der Norm EN 50295 festgeschrieben. Eine Erweiterung fand im August 1998 durch die Spezifikation 2.1 statt. In dieser Erweiterung wurde die „Slave"-Anzahl von 31 auf 62 erhöht [45] und ein Standardverfahren zur Übertragung von Analogwerten, aufgrund der sehr kurzen Telegramme, definiert [25]. Die Spezifikation 2.1 ist abwärtskompatibel und gleichzeitig mit der vorherigen gültig. Mit dem TÜV- und

[23] Manchmal wird dieser Bus auch „ITS Data Bus" genannt.

BIA-zertifizierten "Safety-at-Work“-Konzept darf der ASI-Bus seit ca. zwei Jahren in sicherheitskritischen Systemen der Kategorie 4 nach EN 954-1 eingesetzt werden (siehe 8.8) [40] [25]. Dieses Konzept sieht einen so genannten Sicherheitsmonitor am Bus vor, der die Anlage gegebenenfalls in den sicheren Zustand schaltet, vorausgesetzt ein solcher Zustand existiert.

Funktionsprinzip

Der „Master“ fragt die „Slaves“, nach aufsteigenden Adressen geordnet, in einer Runde ab. Aufgrund deren selbstständiger Wiederholung, stellt sich beim ASI-Bus ein impliziter Zyklus ein. Über einen Automatismus entfernt der „Master“ defekte oder abgesteckte „Slaves“ aus der Runde. Mit neu hinzugekommenen „Slaves“ verhält es sich umgekehrt. Dadurch gestattet der ASI-Bus eine unkomplizierte, zügige Inbetriebnahme und Wartung. Die Bitdauer ist mit 6 µs festgelegt, woraus sich eine Baudrate von knapp 167 kBit/s errechnet. Mit 31 „Slaves“ ergibt das eine durchschnittliche Zykluszeit um 5 ms und mit 62 „Slaves“ um 10 ms [24] [25]. Für industrielle Prozesse mit Zeitkonstanten ab 50 ms ist das hinreichend. Sensor oder Aktor bedeutet beim ASI-Bus nicht gleichzeitig „Slave“, da die Spezifikation den Anschluss von bis zu vier binären Sensoren und/oder Aktoren pro „Slave“ vorsieht. Theoretisch können damit 124 (alte Spez.) bzw. 248 (neue Spez.) binäre Sensoren und/oder Aktoren pro ASI-Bus vernetzt werden. In der Praxis reduzieren sich jedoch diese Werte, durch analoge und intelligente Sensoren und Aktoren.

Bei der APM-Technik wird der Bitstrom nach dem Manchester-Verfahren codiert, weshalb der ASI-Bus mit einer synchronen Übertragung arbeitet. Nach dieser Codierung besteht der Bitstrom aus 3 µs langen Halbbits. Diese werden als speziell geformte Stromimpulse, mit einer Höhe von ca. 60 mA, auf den Bus eingeprägt. An einer systemweiten Induktivität erzeugen sie per Differentation sin²-förmige Spannungsimpulse, mit einer Amplitude von ca. 2 V. Da die so genannte ASI-Spule[24] und das Netzgerät in Reihe geschaltet sind, addieren sich die Spannungsimpulse und die 24 V_{DC}-Versorgungsspannung. Auf diesem Weg erreichen die Daten auf den Stromversorgungsleitungen alle Teilnehmer, die beim ASI-Bus ausschließlich parallel geschaltet sind. Für die Mittelwertfreiheit des Datensignals wechselt die Polarität der Spannungsimpulse fortlaufend. Dafür ist es jedoch notwendig, bei gleichen benachbarten Impulsen jeweils den zweiten zu unterdrücken. In dem Manchester-codierten Datenstrom passiert das, wenn zwei aufeinanderfolgende Bits unterschiedlich sind. Deshalb wird der erste Impuls im zweiten Bit nicht gesendet [43].

Für die Regeneration der Daten genügt im Prinzip ein Filter, Komparator und Manchester-Decoder. Der Filter kann schmalbandig sein, weil das Frequenzspektrum eines sin²-Impulses schnell abklingt (siehe 8.5). Aus diesem Grund ist die APM-Technik störfester als die bei Sensor/Aktorbussen häufig angewandte, unmodulierte Übertragung. Des Weiteren sinken die Störabstrahlung und Leitungsreflexionen, so dass der ASI-Bus ohne Schirmung, verdrillte Leitungen und Terminierungswiderstände auskommt – Letzteres ermöglicht die Baumtopologie. Üblicherweise erfolgt die Verdrahtung mit dem genormten, gelben ASI-Kabel. Der Anschluss ist denkbar einfach. Ein Teilnehmer wird an das ASI-Kabel „aufgeschnappt“, wobei Dorne das Kabel durchdringen und den elektrischen Kontakt herstellen. Eine Unsymmetrie im rechteckähnlichen Querschnitt des ASI-Kabels verhindert eine Verpolung. Die Strombelastbarkeit der 1,5 mm² starken Drähte ist mit 8 A_{DC} spezifiziert. Wegen der zulässigen Netzausdehnung von bis zu 100 m (ohne „Repeater“), wird jedoch ein Maximalstrom von 2 A_{DC} empfohlen. Außerdem sollte der Strombedarf eines „Slaves“ unter 120 mA_{DC} liegen. Da viel mehr Sensoren als Aktoren verbaut werden, und Sensoren i.A. deutlich weniger Strom verbrauchen, stellen diese Restriktionen kein Problem in der Praxis dar.

[24] Die ASI-Spule ist stromkompensiert. Dadurch wird sie nicht vormagnetisiert und kommt mit einem kleinen Kern aus.

Telegramm

Ein ASI-Telegramm besteht aus einem „Master“-Aufruf, gefolgt von einer „Master“-Pause, einer „Slave“-Antwort und einer abschließenden „Slave“-Pause (Bild 2.2). Der „Master“-Aufruf und die „Slave“-Antwort haben Ähnlichkeit mit einem UART-„Frame“. Sie beginnen mit einem „0“-Startbit und enden mit einem „even“ Paritybit, gefolgt von einem „1“-Stoppbit. Nach dem Startbit sendet der „Master“ ein Steuerbit, fünf Adressbits (A4..A0) und fünf Informationsbits (I4..I0). Der „Slave“ dagegen sendet nur vier Informationsbits (I3..I0). Daten werden in den Bits I3..I0 übertragen (pro Sensor oder Aktor ein Bit), und zwar auch vom „Master“. Das Bit I4 wird für eine Umschaltung zwischen Parametern (1) und Daten (0) benötigt. In der neuen Spezifikation kann der „Master“ nur noch drei Datenbits verschicken, weil für die Verdopplung des Adressraums das Informationsbit I3 verwendet wird.

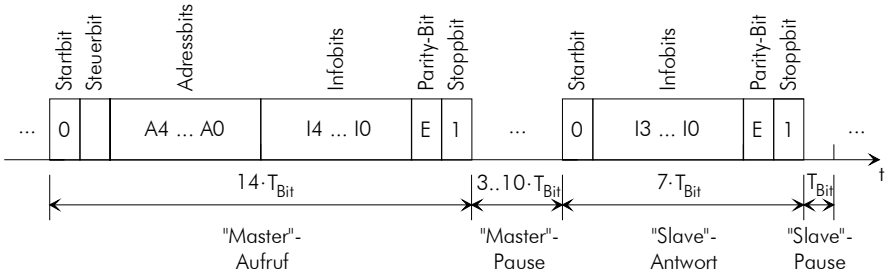


Bild 2.2: ASI-Telegramm

Mit den fünf (sechs) Adressbits A4..A0 (A4..A0 + I3) wählt der „Master“ einen der 31 (62) möglichen „Slaves“ aus. Kombinatorisch lassen sich 32 (64) Adressen darstellen. Adresse 0 (0 und 63) ist jedoch für Service- und Managementzwecke reserviert. Das Steuerbit im „Master“-Aufruf dient zur Auswahl zwischen Kommandos (1) oder Daten bzw. Parametern (0). Die „Master“-Pause beträgt 3..10 Bitzeiten, wird i.A. aber nicht länger als fünf Bitzeiten[25]. Sehr kurz und in der Länge fest ist die „Slave“-Pause mit einer Bitdauer. Insgesamt ergibt das eine Telegrammlänge von 25..32 Bitzeiten.

Zeitverhalten

Der „Master“ kommuniziert mit den „Slaves“ rundenweise und zyklisch (siehe Bild 2.3). Eine Runde besteht aus einer Reihe dicht an dicht liegender ASI-Telegramme. Das erste Telegramm gilt dem „Slave“ mit der niedrigsten Adresse. Sofern der „Slave“ mit der Nummer eins vorhanden ist und funktioniert, ist das die Adresse eins.

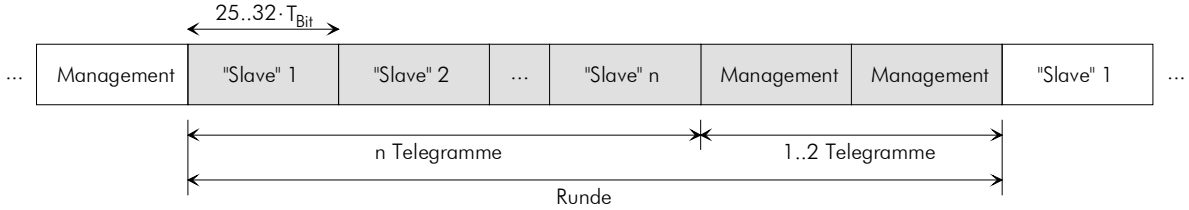


Bild 2.3: ASI-Zyklus

[25] Mehr als fünf Bitzeiten treten meistens nur mit „Repeatern“ auf.

Danach folgen die Telegramme für die „Slaves“ mit den höheren Adressen, in aufsteigender Reihenfolge. Wenn der „Slave“ mit der höchsten, vorhandenen Adresse erreicht ist, schließen ein bis zwei Management-Telegramme die Runde ab. Anschließend beginnt das Ganze, in der Regel ohne Pause, von vorne. Das erste Management-Telegramm ist obligatorisch. Es trägt die Bezeichnung Aufnahmetelegramm, weil der „Master“ damit neue „Slaves“ per Adress-„Scan“ sucht – pro Runde eine Adresse. Mit dem zweiten optionalen Management-Telegramm, dem so genannten Kommandotelegramm, kann er neue und alte „Slaves“ parametrisieren oder konfigurieren.

Da weder die Rundelänge, noch die Telegrammpausen konstant sind, kann die Zykluszeit beim ASI-Bus variieren. Unter der Annahme von n funktionierenden „Slaves“, keiner Baudratentoleranz und keinen Rundenpausen, beträgt das Intervall für die Zykluszeit:

$$T_{Zyklus} = \left[25(n+1)..32(n+2)\right] \cdot T_{Bit} \qquad \text{für } n = 1..31 \text{ bzw. } 62$$

Formel 2.1: Zykluszeit beim ASI-Bus

Mit $T_{Bit} = 6\ \mu s$ und $n = 1$ errechnet sich die kleinstmögliche Zykluszeit zu 0,3 ms. Die größtmögliche Zykluszeit hingegen beträgt für $n = 31$ knapp 6,34 ms bzw. für $n = 62$ knapp 12,29 ms – analoge Signale werden, durch die notwendige Zerlegung, um den Faktor zwei bis drei langsamer abgetastet. Über den Zyklus-„Jitter“ kann man allgemein nur grobe Aussagen machen. Im Normalfall fehlt das zweite Management-Telegramm. Es werden ja nicht ständig neue „Slaves“ angeschlossen oder vorhandene umkonfiguriert. Somit bestimmen die Faktoren 25 und 32 in Formel 2.1 den Zyklus-„Jitter“ im Normalfall. Bezogen auf den Mittelwert von 28,5, beträgt dieser durchschnittlich ca. ± 12,5 %. Für einfache Anwendungen spielt das keine Rolle, unter Umständen jedoch für Abtastregelungen (Regelgüte) u.ä.

Bei fehlerhaften Telegrammen steigt die Zykluszeit durch sofortige Wiederholungen an. Der „Master“ erkennt fehlerhafte Telegramme an einer verfälschten „Slave“-Antwort, einer fehlenden „Slave“-Antwort oder einer zu langen „Master“-Pause. Durch die Begrenzung auf eine Wiederholung pro Runde und „Slave“, kann die Zykluszeit aber nicht beliebig lang werden. Die min. Verlängerung entsteht bei nicht antwortenden „Slaves“ durch zwei „Master“-Aufrufe. Sie beträgt pro „Slave“: $\Delta T_{min} = [2\cdot(14 + 10) - 32]\cdot T_{Bit} = 16\cdot T_{Bit}$. Wenn die „Slaves“ hingegen falsch antworten, werden zwei komplette Telegramme initiiert. Dadurch wird die Verlängerung am größten. Pro „Slave“ kann sie im „Worst-Case“ $\Delta T_{max} = [2\cdot 32 - 25]\cdot T_{Bit} = 39\cdot T_{Bit}$ betragen. Unter der Annahme von durchschnittlich $28{,}5\cdot T_{Bit}$ langen Telegrammen und einer durchwegs gestörten Kommunikation, besteht damit die Möglichkeit, dass die Zykluszeit um 56,1..136,8 % länger wird. Diese Betrachtung ist zwar theoretischer Natur, sie zeigt aber das Realzeitproblem des ASI-Busses im Fehlerfall.

Es kann noch schlimmer kommen. Wenn der „Master“ in drei aufeinanderfolgenden Runden mit einem „Slave“ nicht korrekt kommunizieren kann (sechs Versuche), so wird dieser „Slave“ als defekt angenommen und aus dem Zyklus automatisch entfernt. Dieser nicht seltene Fall führt also erst zu einer kurzfristigen Erhöhung und dann zu einer langfristigen Erniedrigung der Zykluszeit. Aufgrund der impliziten Arbeitsweise, kann man daran leider nichts ändern (z.B. einfügen und anpassen einer Rundenpause). Deshalb ist der ASI-Bus nur bedingt realzeitfähig.

Effizienz

Die Telegrammeffizienz liegt, je nach „Master“-Pause, zwischen $8/32 = 0{,}25$ und $8/25 = 0{,}32$ (alte Spezifikation EN 50295) bzw. $7/32 \approx 0{,}22$ und $7/25 = 0{,}28$ (neue Spezifikation 2.1). Hierbei wird an-

genommen, dass alle acht bzw. sieben Informationsbits im „Master“-Aufruf und in der „Slave“-Antwort genutzt werden. Wegen der Management-Telegramme und der Bandbreitenverdopplung durch die Manchester-Codierung, ist die Gesamteffizienz jedoch ein ganzes Stück geringer. Für den fehlerfreien Fall berechnet sie sich wie folgt:

$$\eta_{31} = \frac{1}{2} \cdot \left[\frac{8 \cdot n}{32 \cdot (n+2)} \;..\; \frac{8 \cdot n}{25 \cdot (n+1)} \right] \qquad \text{für } n = 1..31$$

$$\eta_{62} = \frac{1}{2} \cdot \left[\frac{7 \cdot n}{32 \cdot (n+2)} \;..\; \frac{7 \cdot n}{25 \cdot (n+1)} \right] \qquad \text{für } n = 1..62$$

Formel 2.2: Effizienz des ASI-Busses

$\eta_{31} = f(n)$ gilt für die alte Spezifikation und $\eta_{62} = f(n)$ für die neue Spezifikation. n ist die Variable für die Anzahl der „Slaves“. Der Faktor ½ berücksichtigt die Bandbreitenverdopplung durch die Manchester-Codierung. Bild 2.4 enthält eine graphische Auswertung der beiden Formeln. Für die alte Spezifikation gelten die durchgezogenen Kurven und für die neue Spezifikation die gestrichelten. Man erkennt, dass die Effizienz des ASI-Busses bis ca. fünf „Slaves“ rasch ansteigt. Ab gut zehn „Slaves“ liegt die Effizienz in einem vernünftigen Bereich (≈ 13 % bzw. ≈ 10 %). Wegen dieser moderaten Werte, liegt die Nettodatenrate bei ca. 45 kBit/s bzw. 35 kBit/s (0,14/3 µs bzw. 0,10/3 µs).

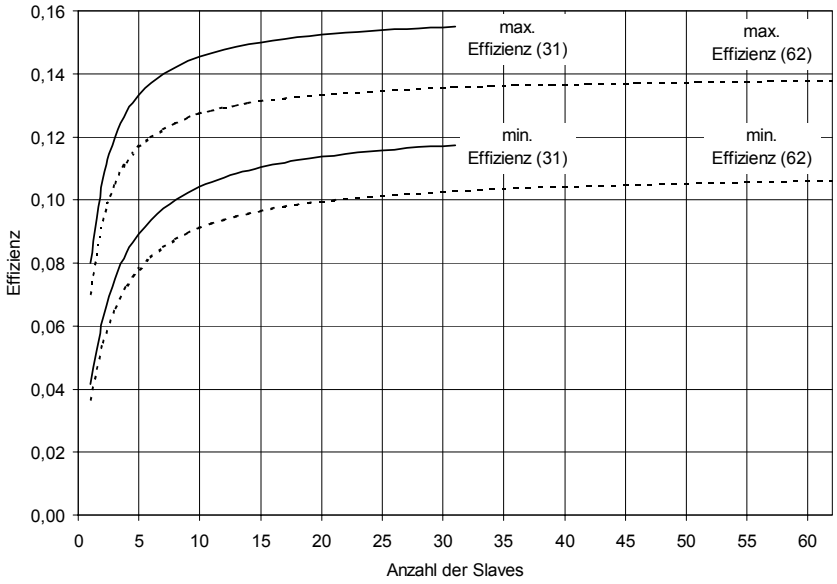


Bild 2.4: Effizienz des ASI-Busses

Verlässlichkeit

Für die Datensicherung wird ein Set von sieben Regeln angewendet. Geprüft werden zeitliche Restriktionen, logische Abläufe und das Paritybit. So muss z.B. der erste Impuls eines Telegramms negativ und der letzte positiv sein. Auf einen positiven Impuls folgt immer ein negativer Impuls und umgekehrt. Außerdem müssen Impuls- und Telegrammpausen in gewissen zeitlichen Schranken lie-

gen. Aufgrund der UND-Verknüpfung dieser Regeln, lassen sich bis zu zwei Impulsfehler (Halbbitfehler) sicher erkennen und mit einer Wahrscheinlichkeit von 99,9999 % drei und vier Fehler. Das schmalbandige Frequenzspektrum und die kurzen Telegramme wirken sich zudem günstig auf die Störsicherheit aus. Eine Simulation mit gleichverteilten Bitfehlern hat ergeben, dass die Restfehlerwahrscheinlichkeit, für eine mittlere Bitfehlerwahrscheinlichkeit unter 10^{-3}, kleiner als 10^{-12} ist[26]. Nach DIN 19244 (siehe Abschnitt 8.7) erfüllt der ASI-Bus unterhalb dieser Grenze die Anforderungen der höchsten Datenintegritätsklasse (I3: kritische Informationsübertragung) [24]. Für mittlere Bitfehlerwahrscheinlichkeiten von $10^{-3}..3\cdot10^{-2}$, fällt der ASI-Bus in der Datenintegritätsklasse I2 (spontane Übertragung). Oberhalb von $3\cdot10^{-2}$ (≈ 5000 Bitfehler/s) bricht die Kommunikation zusammen – der „Master" geht davon aus, dass alle „Slaves" defekt sind. Da so hohe Fehlerraten in der Regel durch „Bubbling-Idiots" entstehen, besitzt jeder ASI-„Slave" eine so genannte „Jabber-Inhibit"-Einheit[27], die „Bubbling-Idiots" rechtzeitig vom Bus trennen.

Trotz der niedrigen Restfehlerwahrscheinlichkeit, stellt die min. Codedistanz $d_{min} = 3$ (Halbbitfehler) eine gewisse Einschränkung dar. Offenbar gibt es Codewörter mit großer und kleiner Codedistanz, wodurch der Coderaum unausgewogen genutzt ist. Von den Codewörtern mit kleiner Codedistanz kann es nicht viele geben, sonst wäre die Restfehlerwahrscheinlichkeit höher. Dennoch bilden sie eine Schwachstelle. Wenn man den entsprechenden Zustand einstellt, d.h. die entsprechenden Telegramme versendet, so kann der im gesamten robuste ASI-Bus relativ leicht gestört werden. Mit einer gleichmäßigeren Ausnutzung des Coderaumes entstünde diese Achillesferse nicht.

2.3.2 ISO 9141-Bus

Allgemeines

Mit DIN ISO 9141 wurde 1989 ein einfaches, serielles Bussystem für die Diagnose in Straßenfahrzeugen definiert [58]. Der ISO 9141-Bus überträgt Daten asynchron in UART-„Frames" und arbeitet nach dem „Master/Slave"-Prinzip. Die Topologie ist linienförmig mit Ausdehnungen bis 40 m. Ein richtiger Sensor/Aktorbus oder gar Feldbus ist der ISO 9141-Bus jedoch nicht. Dafür spezifiziert seine Norm zu wenig. Seine heutige Bedeutung liegt im „Physical-Layer". Er ist einfach, kostengünstig und wird in vielen Bussystemen für die Diagnose von Kfz-Steuergeräten u.ä. verwendet (OBD-II, J1708, Bean, J1850, CCD...)[28]. Auch einige der neuen Sensor/Aktorbusse, wie z.B. der LIN- oder PPC-Bus, machen davon Gebrauch. Neben der Bezeichnung ISO 9141-Bus sind ISO-K-Bus, K-Line und Diagnosebus gebräuchlich[29].

Funktionsprinzip

Eine Kommunikation beginnt beim ISO 9141-Bus mit einer Initialisierungsphase, gefolgt von einem Synchronisations-„Frame" und mindestens einem Schlüsselwort à zwei „Frames". Danach beginnt die Übertragung der Daten, bei einer Baudrate von 10 Bit/s bis 10 kBit/s. Initiator ist in vielen Fällen ein so genanntes Diagnoseprüfgerät. Es sendet entweder eine Adresse mit 5 Bit/s, oder für ungefähr 1,8 s eine logische „0". Außerdem sieht die Norm eine Notlösung für die Initialisierung vor. Dazu muss man die Datenleitung auf Masse legen und die Zündung einschalten. Gut 2 s später ist die Initialisierungsphase dann abgeschlossen.

---

[26] Im Durchschnitt vergehen damit mehr als 12 Jahre zwischen zwei unerkannten Fehlern.

[27] Ein gängiges Synonym ist „Bus-Guardian".

[28] Der SAE-Standard ODB-II enthält das neue Diagnoseprotokoll Keyword 2000.

[29] Die Norm definiert eine K- und L-Datenleitungen. Von beiden muss die K-Datenleitung immer vorhanden sein.

Der „Physical-Layer“ besteht aus einer Eindrahtleitung und „Open Collector“-Leitungstreibern („Wired-AND“). Über ihn werden die serialisierten Informationen im NRZ-Format übertragen. Eine logische „0“ wird durch Einprägen einer Spannung von max. $0{,}2 \cdot U_{Batt}$ gesendet. Zur Sicherheit akzeptiert der Empfänger sie bis $0{,}3 \cdot U_{Batt}$. Analog gelten für eine logische „1“ die Werte $0{,}8 \cdot U_{Batt}$ und $0{,}7 \cdot U_{Batt}$. Als Betriebsspannung U_{Batt} sind 12 V (Kfz) und 24 V (LKW) definiert. Die Signalflanken müssen innerhalb von $0{,}1 \cdot T_{Bit}$ ansteigen bzw. abfallen. Für eine diskrete Busanschaltung reichen ein NPN-Transistor, ein paar Widerstände und ein Schmitt-Trigger aus. Vorwiegend werden jedoch Komplett-ICs verwendet. Sie besitzen, neben dem Sender und Empfänger, einen Kurzschluss- und Überspannungsschutz (z.B. MC 33290 von Motorola oder TLE 6258 von Infineon). Die Leitungslänge wird indirekt über die Buskapazität definiert: $f_{Bit} \leq 10^{-4} Hz \cdot F / C_{ges}$. Aus praktischen Erfahrungen weiß man jedoch, dass Netzausdehnungen bis 40 m und Baudraten bis 20 kBit/s (außerhalb der Spezifikation) unproblematisch sind.

Telegramme

Für den „Data-Link-Layer“ spezifiziert die Norm ISO 9141 ein UART-„Frame“ mit einem Startbit, sieben Datenbits, einem ungeraden Paritybit und einem Stoppbit. Damit beträgt die „Frame“-Länge zehn Bits. Zwischen zwei „Frames“ muss eine Pause von 0,2 ms bis 1,2 s liegen. Mit Ausnahme des Initialisierungstelegramms, nachdem eine Pause von bis zu 2 s folgen darf, beruhen alle anderen Telegramme des ISO 9141-Busses auf diesen Regeln. Das Initialisierungstelegramm (5 Bit/s) enthält eine 7 Bit-Adresse für die Auswahl eines Steuergeräts. Theoretisch können damit 128 Steuergeräte unterschieden werden. Aufgrund der Buskapazität, wird jedoch ein Maximalwert von 16 Steuergeräten pro Bus empfohlen [52]. Nach dem Initialisierungstelegramm folgt ein Synchronisationstelegramm. Es übermittelt den Steuergeräten die Baudrate mit einem alternierenden Bitmuster. Anschließend folgen firmenspezifische Schlüsselwort- und Datentelegramme.

Zeitverhalten

Das Zeitverhalten liegt, wegen den sehr unterschiedlich langen Pausen, im oberen ms- bis s-Bereich. Bei einer Wartung spielt das keine Rolle. Für technische Prozesse ist der ISO 9141-Bus i.A. jedoch zu reaktionslangsam. Mit entsprechenden Einschränkungen der Pausenzeiten lässt sich dieses Manko beseitigen. Allerdings hat das dann wenig mit ISO 9141 zu tun. Dennoch wird es, z.B. in BMW-Cockpits, angewendet.

Effizienz

Die reine „Frame“-Effizienz des ISO 9141-Busses hat einen Wert von 7/10 = 0,7. Wegen den Pausen, liegt die Gesamteffizienz jedoch weit darunter. Für die folgende Abschätzung wird die max. Baudrate des ISO 9141-Busses von 10 kBit/s angenommen. Kleinere Baudraten machen heutzutage keinen Sinn mehr. Mit 10 kBit/s dauern die „Frames“ 1 ms, wovon die Datenbits 0,7 ms belegen. Dauerhafte Pausenzeiten von 0,2 ms und 1,2 s sind unrealistisch. Vernünftig sind durchschnittliche Pausenzeiten von 3..6 ms, woraus sich eine Effizienz von ca. 10..17 % für den ISO 9141-Bus errechnet.

Verlässlichkeit

Die Fehlersicherung des ISO 9141-Busses besteht aus einem Paritybit. Es bietet einen Fehleraufdeckungsgrad von 0,5 und eine min. Codedistanz $d_{min} = 2$. Unter der Annahme von statistisch unabhängigen Bitfehlern und einem BSC-Kanal (siehe Abschnitt 8.2), beträgt die Wahrscheinlichkeit für verfälschte, nicht erkannte Codewörter:

$$P_{Restfehler} = \sum_{e} \binom{8}{e} \cdot P_{Bitfehler}^{e} \cdot (1 - P_{Bitfehler})^{8-e} \qquad \text{für } e = 2, 4, 6, 8$$

Formel 2.3: Restfehlerwahrscheinlichkeit des ISO 9141-Busses

Für eine Quantifizierung benötigt man eine Annahme über die Bitfehlerwahrscheinlichkeit. Aufgrund der asymmetrischen Signalübertragung mit offener, ungeschirmter Eindrahtleitung ist der ISO 9141-Bus störempfindlich. Deshalb wird mit einer relativ hohen Bitfehlerwahrscheinlichkeit $P_{Bitfehler} = 10^{-3}$ gerechnet [69] [100]. Hierbei beträgt die Restfehlerwahrscheinlichkeit: $P_{Restfehler} = 2{,}78 \cdot 10^{-5}$. Nach DIN 19244 (siehe Abschnitt 8.7) erreicht der ISO 9141-Bus damit nur eine Datenintegritätsklasse (I1: Fernmessen), wenn die Daten zyklisch übertragen werden.

2.3.3 LIN-Bus

Allgemeines

LIN („Local Interconnect Network“) ist ein neues, serielles Bussystem für die kostengünstige Vernetzung von Sensoren und Aktoren in Kfzs[30]. Mit dem LIN-Bus sollen die SAE-Klasse A (siehe 8.6) und einfache, mechatronische Anwendungen abgedeckt werden [52] [63]. Beispiele hierfür sind Fensterheber, Zentralverriegelung, Spiegelverstellung, Sitzheizung, Schiebedach usw. (siehe Bild 2.5). Die Anforderungen bezüglich Sicherheit, Geschwindigkeit, Realzeitfähigkeit und Zuverlässigkeit sind beim LIN-Bus eher gering. In erster Linie spielen die Kosten eine Rolle. Sie müssen unter denen der konventionellen Verdrahtungstechnik liegen, damit neben der Innovation auch Kosten gespart werden können. Da es in der Kfz-Branche um sehr hohe Stückzahlen geht, zählen bereits ein paar Cents. Außerdem haben die Kfz-Hersteller ein Rechtfertigungsproblem für Mehrkosten. Wie in Kapitel 1 beschrieben, sind die Kunden i.A. nicht bereit für etwas mehr zu zahlen, das andere Hersteller in konventioneller Technik günstiger anbieten.

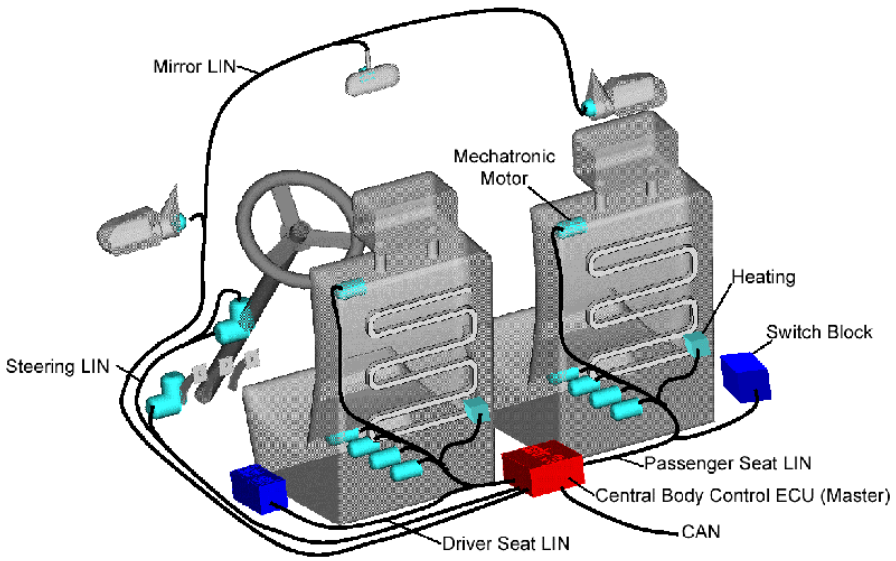


Bild 2.5: Fallstudie für den LIN-Bus [63]

[30] Der LIN-Bus soll ungefähr halb so teuer werden wie der CAN-Bus [52].

An der Ende 1998 gestarteten Entwicklung sind bzw. waren die Firmen Audi, BMW, DaimlerChrysler, Volvo, VW, VCT und Motorola beteiligt. Diese Firmen haben sich mittlerweile zu einem LIN-Konsortium zusammengeschlossen, das für die Spezifikationspflege verantwortlich zeichnet [53]. Momentan ist die Spezifikation 1.2 vom November 2000 gültig [52]. Neben den sonst üblichen Busdefinitionen, findet man dort eine zusätzliche Schnittstellenbeschreibung für Software-Werkzeughersteller, eine „Configuration Language“ und ein „Application Programmer’s Interface“ (API).

Erste LIN-Bausteine sind auf dem Markt bereits erhältlich. An „Transceivern“ findet man ICs von Infineon (TLE 6258), Alctel (MTC 30600) und Melexis (TH 806x). Da LIN den „Physical-Layer“ von ISO 9141 verwendet, sind auch ICs von z.B. Motorola (MC 33290) verwendbar. Auf der µC-Seite bietet STMicroelectronics die ST 72x-Familie an, Motorola den MC 68HC908EY8 und MC 9S12DP256, und Microchip die PIC 16C43x-Familie, um die Wichtigsten zu nennen. Mit einer Zunahme der Vielfalt ist zu rechnen, da viele Automobilhersteller den LIN-Bus favorisieren. In der Serie wird er aber noch nicht eingesetzt.

Funktionsprinzip

In der LIN-Spezifikation werden die OSI-Schichten 1 („Physical-Layer“) und 2 („Data-Link-Layer“) definiert. Der „Physical-Layer“ entspricht dem des ISO 9141-Busses: Asymmetrische Übertragung der Bits im NRZ-Format über eine ungeschirmte Eindrahtleitung mit einem Pegelhub von ca. 12 V und „Open Collector“-Technik. Letzteres erzeugt einen dominanten (niederohmigen) und rezessiven (hochohmigen) Buspegel. Ein logische „1“ wird rezessiv bei ca. 12 V übertragen und eine logische „0“ dominant bei ca. 0 V. In dieser auch „Wired-AND“ genannten Technik wird ein „Pull-Up“-Widerstand benötigt. Er ist auf alle Teilnehmer eines Netzes verteilt. Im „Master“ beträgt sein Wert 1 kΩ und in den „Slaves“ 30 kΩ. Explizite Abschlusswiderstände werden nicht benötigt. An der linienförmigen Topologie des LIN-Busses können bis zu 63 Teilnehmer angeschlossen werden. Empfohlen ist jedoch eine max. Anzahl von 16. Die Baudrate kann zwischen 1 und 20 kBit/s betragen, wobei 2,4 kBit/s, 9,6 kBit/s und 19,2 kBit/s zu bevorzugen sind[31]. Bei der Netzausdehnung ist eine Obergrenze von 40 m definiert.

Im „Data-Link-Layer“ sieht der LIN-Bus eine „Master/Slave“-Arbitrierung vor. Für den Datentransport stellt er ein Telegramm mit variabler Länge zur Verfügung. Es setzt sich, bis auf eine Ausnahme, aus Standard-UART-„Frames“ zusammen. Im Gegensatz zu klassischen „Master/Slave“-Bussen, die mit Teilnehmeradressen arbeiten, verwendet der LIN-Bus eine Nachrichtenadressierung. Dazu versendet der „Master“ im vorderen Teil eines Telegramms einen so genannten „Identifier“. Der „Identifier“ unterscheidet 64 Nachrichten, von denen in der Regel nur wenige für einen „Slave“ interessant sind. Deshalb sieht der LIN-Bus, so wie der CAN-Bus, eine Nachrichtenfilterung in den Teilnehmern vor. Die Reaktionen auf eine Nachricht sind: Nichtstun, Zuhören, Senden, Konfiguration, „Sleep“-Zustand, und Benutzeraktionen. Ein LIN-spezifisches „Task“-Konzept verkoppelt jede Nachricht mit einer „Slave-Task“ und erlaubt Punkt-zu-Punkt- und „Multicast“-Verbindungen[32]. „Slave-Tasks“ können auf die Knoten eines Netzes beliebig verteilt werden – auch auf den „Master“. Eine so genannte, einmalige „Master-Task“ macht aus einem Knoten den „Master“. Sie initiiert Nachrichten durch Versenden eines „Headers“, der den Nachrichten-„Identifier“ enthält.

---

[31] Baudraten unterhalb von 1 kBit/s sind wegen „Timeout“-Restriktionen nicht möglich.

[32] Auf den ersten Blick ähnelt der LIN- dem CAN-Bus. Beim CAN-Bus kann jedoch jeder Teilnehmer den Bus arbitrieren (CSMA/CA), beim LIN-Bus nicht. Des weiteren sind beim CAN-Bus die Nachrichten priorisiert, während dessen sie beim LIN-Bus gleichberechtigt sind. Und schließlich arbeitet der CAN-Bus bitorientiert und der LIN-Bus zeichenorientiert.

Für die Uhrensynchronisation enthält der „Header“ ein spezielles Bitmuster. Die „Slaves“ leiten daraus den „Master“-Takt ab und kalibrieren so laufend ihre Uhren und „Timer“. Aus diesem Grund kann man beim LIN-Bus auch „low-cost“ µCs mit internem Oszillator („On-Chip“) verwenden. Die Baudratenabweichungen dürfen im Betrieb ±2 % betragen und beim Hochfahren ±15 %; Bezug ist der „Master“. Allerdings lässt die LIN-Spezifikation offen wie ein hochlaufender „Slave“ mit diesem Fehler die richtige Baudrate findet. Des Weiteren können „On-Chip“-Oszillatoren einen deutlich höheren Frequenzfehler aufweisen (siehe Abschnitt 8.4). Aus diesen Gründen wird vermutet, dass ein LIN-„Slave“ mit „low-cost“ µC so ähnlich „bootet“ wie ein entsprechender TTP/A-„Slave“. Dort sucht ein Algorithmus[33] die richtige Geschwindigkeit durch Kombinieren des Baudraten- und Kalibrierregisters („On-Chip“-Oszillator).

Telegramm

Sämtliche Daten bzw. Nachrichten werden beim LIN-Bus in einem so genannten „Message-Frame“ übertragen. Den Aufbau zeigt Bild 2.6 [52]. Mit Ausnahme des „Sync. Break“-Teils, ist die atomare Dateneinheit ein UART-„Frame“ mit zehn Bits (ein Startbit, acht Datenbits, ein Stoppbit). Im vorderen „Header“-Teil sendet immer der „Master“ und im hinteren „Response“-Teil entweder der „Master“ oder ein „Slave“. Je nach Verbindungstyp, können im „Response“-Teil ein oder mehrere Teilnehmer zuhören. Zwischen dem „Header“- und „Response“-Teil existiert eine Pause, der so genannte „In-Frame-Space“.

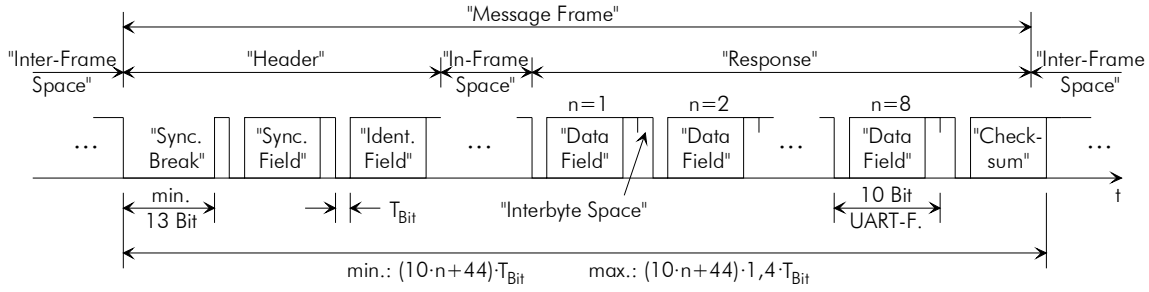


Bild 2.6: LIN-Telegramm bzw. -„Message-Frame“

Ein „Message-Frame“ beginnt mit einer Synchronisationsphase, in der die „Slaves“ ihre Uhren kalibrieren. Dazu sendet der „Master“ in dem „Sync. Break“-Teil min. dreizehn logische „1en“, gefolgt von min. einer logischen „0“ (Stoppbitpegel)[34]. Da in einem LIN-UART-„Frame“ max. neun „1en“ hintereinander auftreten können, ermöglicht diese Methode eine klare Unterscheidung zwischen der Daten- und Synchronisationsphase. Aufgrund der Uhrenfehler schalten die „Slaves“ jedoch erst nach elf „1en“, gemessen mit ihrer Uhr, auf Synchronisation um. Diese beginnt mit dem nächsten „Frame“, dem „Sync. Field“. In ihm sendet der „Master“ das Datenbyte 0x55. Zusammen mit dem Start- und Stoppbit entsteht die alternierende Bitfolge („0101010101“). Aus ihr kann ein „Slave“ auf mannigfache Art und Weise den „Master“-Takt bestimmen. Um den Messfehler gering zu halten, sollte man das Messintervall groß wählen. Empfohlen wird der Zeitraum zwischen der Startbitflanke und der fallenden Flanke zu Beginn von Bit sieben. Das ist zwar nicht das größtmögliche Messintervall, aber ein sehr klug gewähltes. Es werden acht „Master“-Bitzeiten vermessen, wodurch die notwendige Division mittels drei einfacher „Shift Right“-Operationen ausgeführt werden kann.

[33] Im einfachsten Fall ist das eine systematische Suche.

[34] Für einen normalen UART ist dieses „Frame“ schlichtweg zu lang. Deshalb benötigt zumindest der „Master“ mehr als einen UART (z.B. SPI), wenn man nicht ständig die Baudrate ändern möchte.

Den letzten Teil im „Header“ bildet das „Identifier Field“ (Bild 2.7). Es enthält sechs „Identifier“-Bits (ID0..ID5) und zwei Paritybits (P0, P1). Mit den „Identifier“-Bits können bis zu 64 verschiedene Nachrichten codiert werden. Da die Bits ID4 und ID5 gleichzeitig die Anzahl der Datenbytes im „Response“-Teil spezifizieren („00“ und „01“: zwei Bytes, „10“: vier Bytes, „11“: acht Bytes), kann man die Länge des „Response“-Teils nicht frei wählen. Außerdem stehen nicht alle 64 Nachrichten zur Verfügung. Die „Identifier“ 0x3C und 0x7D sind für so genannten „Command-Frames“ und 0xFE und 0xBF für so genannten „Extended-Frames“ reserviert. „Command-Frames“ dienen zu Servicezwecken und „Extended-Frames“ für benutzerspezifische Nachrichten, sowie zukünftige LIN-Erweiterungen. In beiden Fällen werden Datenbytes im „Response“-Teil belegt. Weitere Details enthält die Spezifikation [52].

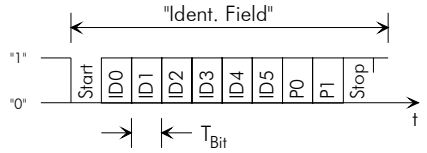


Bild 2.7: LIN-„Identifier Field“

Für den Datentransport stehen bis zu acht „Data-Fields“ im „Response“-Teil eines LIN-Telegramms zur Verfügung. Im Gegensatz zu den UART-„Frames“ im „Header“-Teil, darf zwischen ihnen eine kleine Pause existieren, der so genannte „Interbyte-Space“. Weder „Interbyte-Space“, noch „In-Frame-Space“ sind in ihrer Dauer definiert. Allerdings gibt die Spezifikation eine min. und max. Telegrammlänge vor, woraus man eine Obergrenze für die Summe der Pausenzeiten berechnen kann. Die max. Telegrammdauer beträgt $1{,}4\cdot(10\cdot n + 44)\cdot T_{Master,\,Bit}$ und die min. Telegrammdauer $(10\cdot n + 44)\cdot T_{Master,\,Bit}$ (Anzahl der Datenfelder n = 2, 4, 8). Hieraus folgt als Obergrenze für die Summe der Pausenzeit: $\Sigma T_{Pause} = 0{,}4\cdot(10\cdot n + 44)\cdot T_{Master,\,Bit}$.

Zeitverhalten

Mit diesen Angaben lassen sich Zeitschranken für die Telegramme berechnen: Das Minimum entsteht für $f_{Master,\,Bit} = 20$ kBit/s, $n = 2$ und $\Sigma T_{Pause} = 0$. Es hat einen Wert von $T_{min} = (10\cdot n + 44)\cdot T_{Master,\,Bit} = 64$ Bit·50 µs/Bit = 3,2 ms. Für das Maximum gelten die Parameter $f_{Master,\,Bit} = 1$ kBit/s, $n = 8$, und $\Sigma T_{Pause} = 0{,}4\cdot(10\cdot n + 44)\cdot T_{Master,\,Bit}$, woraus der Wert $T_{max} = 1{,}4\cdot(10\cdot n + 44)\cdot T_{Master,\,Bit} = 173{,}6$ Bit· 1 ms/Bit = 173,6 ms resultiert. Oszillator- bzw. Baudratentoleranzen sind in dem Faktor 1,4 enthalten. Die min. Telegrammdauer bildet ein Infimum für Abtastperioden. Zwar ist der LIN-Bus an den CAN-Bus angelehnt, doch aufgrund der „Master/Slave“-Arbitirierung können die „Slaves“ nicht selbstständig auf Ereignisse reagieren. Sie müssen warten, bis der „Master“ sie zum Senden auffordert. Deshalb macht beim LIN-Bus nur ein zyklischer Betrieb Sinn. Auf Dauer lässt sich eine Periodendauer von 3,2 ms allerdings nicht einstellen, da die Pausenzeiten variieren dürfen. Sie können für $f_{Master,\,Bit} =$ 20 kBit/s und n = 2 bis zu 0,4·3,2 ms = 1,28 ms pro Telegramm lang werden. Das heißt, die min. Telegrammdauer kann zwischen 3,2 und 4,48 ms schwanken. Unter Berücksichtigung von Rechen- und Latenzzeiten, wird die kleinste Abtastzeit somit auf 5 ms geschätzt.

Nun hat eine Anwendung mit nur einem periodischen Telegramm eher „Benchmark“-Charakter. Aus diesem Grund soll ein praxisrelevantes Zeitverhalten aus dem Beispiel in Bild 2.5 abgeleitet werden. Dort sind vier LIN-Busse mit bis zu sieben Knoten („Seat LIN“) eingezeichnet. Jeder Knoten wird durchschnittlich drei Aufgaben erfüllen; der „Switch Block“ etwas mehr, die Heizung etwas weniger. Bei sieben Knoten macht das 21 Aufgaben, die entsprechend dem „Task“-Konzept an je ein Telegramm gebunden sind. Große Datenmengen werden in diesem Beispiel sicher nicht ausgetauscht,

weshalb von dem kürzesten Telegramm mit zwei Datenbytes ausgegangen wird. Außerdem werden max. Baudrate, zyklischer Betrieb und gleichhäufige Telegramme angenommen. Dadurch ergibt sich eine Abtastzeit bzw. Telegrammperiode von 21·5 ms = 105 ms.

Leider ist der eigentlich notwendige zyklische Betrieb beim LIN-Bus nicht implizit. Er muss in der so genannten „Master-Task", also in der Applikation, programmiert werden. Für eine Fehlerbehandlung gilt das Gleiche [52], weshalb auch Telegrammwiederholungen nicht ausschließbar sind. Aus diesen Gründen kann der LIN-Bus allgemein nur als bedingt realzeitfähig eingestuft werden. Für die anvisierten einfachen Anwendungen reicht das jedoch völlig aus.

Effizienz

Ein LIN-Telegramm kann, je nach Anzahl n der Datenbytes, 1..1,4·(10·n + 44) Bits lang sein. Im „Best Case" (n = 8, min. Telegrammlänge) beträgt die Telegrammeffizienz 8·8 Bit/(10·8 + 44) Bit ≈ 0,52 und im „Worst-Case" (n = 2, max. Telegrammlänge) 2·8 Bit/1,4·(10·2 + 44) Bit ≈ 0,18. Über die Gesamteffizienz kann man allgemein wenig aussagen, weil die Pausen zwischen zwei Telegrammen beliebig lang sein dürfen; nach einer „Idle"-Zeit von $25000 \cdot T_{Bit}$ (≈ 1..3s) schalten die „Slaves" in den „Sleep Mode". Dennoch ist eine Abschätzung für Sensor/Aktoranwendungen möglich. Typisch hierfür sind kleine Datenmengen, weshalb das kürzeste LIN-Telegramm mit zwei Datenbytes ausgewählt wird. Mit den Schwankungen in den Telegrammpausen ergibt das eine Telegrammeffizienz von ca. 0,18..0,25. Aufgrund der undefinierten Pausen zwischen zwei Telegrammen, kann dieser Wert theoretisch nahe an Null herangehen. Relevant ist jedoch eine Regelmäßigkeit beim Lesen der Sensoren und Schreiben der Aktoren. Deshalb, und wegen den zwei Datenbytes pro Telegramm, die für typische Sensor/Aktoranwendungen mehr als genug sind, wird die mittlere Gesamteffizienz auf 10..20 % geschätzt.

Verlässlichkeit

Für die Ermittlung der Restfehlerwahrscheinlichkeit müssen zwei Fälle untersucht werden, da der „Header"- und „Response"-Teil verschieden gegen Übertragungsfehler gesichert sind. Komplizierter ist der „Response"-Teil, weshalb er als Erster analysiert wird. Die Absicherung erfolgt mit einem Prüfbyte am Ende des „Response"-Teils (siehe Bild 2.6). Es wird aus der invertierten Summe der Datenbytes auf dem endlichen Zahlenkörper K = {0..255} gebildet. Die Formel lautet:

$$\overline{CS} = DB_1 \oplus DB_2 \oplus \ldots \oplus DB_8 \qquad GF(256)\,^{35}$$

CS ist die so genannte „Checksum" und $DB_1..DB_8$ die zahlenmäßige Darstellung der Datenbytes. Das Verfahren ist einfach umsetzbar und wird normalerweise beim Programmieren von EPROMs, µCs u.ä. angewandt. Gegenüber einer Block-, Hamming- oder CRC-Codierung, ist es nicht besonders effizient. Außerdem ist die Berechnung der Restfehlerwahrscheinlichkeit relativ schwierig. Wie sich noch zeigen wird, sind die Codedistanzen sehr unregelmäßig. Deshalb kann man keine Fehlergruppen bilden, so wie z.B. beim Paritybit. Dort bleibt jeder geradzahlige Fehler unerkannt, wodurch die Restfehlerwahrscheinlichkeit mit einer Binomialverteilung geschlossen darstellbar ist. Würde man die Binomialverteilung beim dem beschriebenen Additionsverfahren anwenden, so wäre das Ergebnis zu pessimistisch. Unerkannte Fehler können hier nämlich bei 2, 3, 4... Bitverfälschungen auftreten, sie müssen aber nicht. Mit der Binomialverteilung ginge man jedoch davon aus, dass alle diese Bitverfälschungen unerkannt bleiben. Deshalb benötigt man die Anzahl der unerkannten Fehlerkombinationen für jede Bitfehlerzahl. Theoretisch könnte man sie mit einer systematischen Suche rechnergestützt ermitteln. Allerdings liegt hier ein NP-Problem vor, das bei 9·8 Bits $2^{72} \approx 4{,}7 \cdot 10^{21}$ zu untersuchende Fälle produ-

[35] GF(p): endlicher Zahlenkörper („Galois-Field") :mit p Elementen; auch modulo-p Rechnung genannt.

ziert. Selbst ein schneller Rechner könnte bei dieser Menge in keiner vernünftigen Zeit ein Ergebnis liefern.

Glücklicherweise kann die Restfehlerwahrscheinlichkeit aber abgeschätzt werden, zumal eine exakte Berechnung, wegen des unbekannten Kanalmodells, sowieso nicht möglich ist. Im ersten Schritt muss die Entstehung von unerkannten Fehlern geklärt werden. Angenommen man verfälscht in irgendeinem Datenbyte eine „1“ und in einem anderen Datenbyte an gleicher Stelle eine „0“, dann bleibt die Prüfsumme davon unberührt. Gleiches gilt für zwei gleichartige Verfälschungen in MSBs. Durch den Modulo ergibt das in Summe Null. Diese Beispiele beweisen zum einen die min. Codedistanz $d_{min} = 2$. Zum anderen verdeutlichen sie, dass Bitfehler nicht erkennbar sind, wenn sie sich in Summe subtrahieren oder über den Modulo auslöschen. Auf diese Weise lässt sich sehr einfach ein Dreibitfehler konstruieren. Zwei verfälschte „1“ mit einer Wertigkeit 2^0 werden von einer verfälschten „0“ mit der Wertigkeit 2^1 kompensiert. Ähnliche Fälle lassen sich auch für Vier-, Fünf, Sechsbitfehler usw. angeben. Insgesamt beträgt die Anzahl der Fehler dieser Sorte bei 9·8 Bits etwa $1{,}8 \cdot 10^{19}$ – in einem endlichen Zahlenkörper GF(k) wiederholt sich eine Zahl bei n Bits 2^n/k-mal. Außerdem können auch Bitfehlerkombinationen, die das Prüfbyte betreffen unerkannt, bleiben. Eine verfälschte „1“ im LSB des Prüfbytes und im LSB eines Datenbytes ergeben ein gültiges Codewort. Für andere Bitstellen gilt das Gleiche, solange auf die Prüfsumme und einem Datenbyte derselbe positive oder negative Wert addiert wird. Bei Drei-, Vier-, Fünfbitfehler usw. verhält es sich analog. Im Gegensatz zur Anzahl der vorherigen Fehler, fällt die Menge dieser Fehler klein aus.

Bitfehler x	**Kombinationen (x über 72)**	$\mathbf{P_{(Restfehler\|x\ Bitfehler)}}$
0	1	$\approx 9{,}30 \cdot 10^{-1}$
1	72	$\approx 6{,}71 \cdot 10^{-2}$
2	2556	$\approx 2{,}38 \cdot 10^{-3}$
3	59640	$\approx 5{,}57 \cdot 10^{-5}$
4	1028790	$\approx 9{,}61 \cdot 10^{-7}$
5	13991544	$\approx 1{,}31 \cdot 10^{-8}$
6	156238908	$\approx 1{,}46 \cdot 10^{-10}$

Tabelle 2.1: Bedingte Restfehlerwahrscheinlichkeiten bei LIN-Busdaten ($P_{Bitfehler} = 10^{-3}$)

Der zweite Schritt besteht darin, die Fehler auf ihr Gewicht (Auswirkung) hin zu untersuchen. Für die Restfehlerwahrscheinlichkeit spielen auf jeden Fall Zweibitfehler eine Rolle, weil sie am wahrscheinlichsten sind. Ob Drei- oder gar Vierbitfehler von Bedeutung sind, hängt von ihrer Häufigkeit ab. Um das abschätzen zu können, wird eine „Worst-Case“-Betrachtung mit der Binomialverteilung und statistisch unabhängigen Bitfehlern in einem BSC-Kanal (siehe Abschnitt 8.2), durchgeführt. In Tabelle 2.1 sind die bedingten Restfehlerwahrscheinlichkeiten für eine Bitfehlerwahrscheinlichkeit von 10^{-3} gelistet. Erwartungsgemäß ist die bedingte Restfehlerwahrscheinlichkeit für zwei Bitfehler am größten, und zwar um mindestens eine Zehnerpotenz, trotz der wesentlich größeren Anzahl von Drei- und Vierbitfehlern. Damit kann in erster Näherung folgende obere und untere Schranke für die Restfehlerwahrscheinlichkeit angegeben werden: $P_{Rest,\,max} = 1 - P_{(Rest|0\ Bitfehler)} - P_{(Rest|1\ Bitfehler)} \approx 2{,}4 \cdot 10^{-3}$ und $P_{Rest,\,min} = 1 - P_{(Rest|0\ Bitfehler)} - P_{(Rest|1\ Bitfehler)} - P_{(Rest|2\ Bitfehler)} \approx 5{,}7 \cdot 10^{-5}$. Außerdem zeigt Tabelle 2.1 die Wahrscheinlichkeit für eine unverfälschte Übertragung. Sie beträgt bei acht Datenbytes ungefähr 90 %, wenn man zusätzlich die „Ident“-Bits berücksichtigt. Von zehn Telegrammen ist für $P_{Bitfehler} = 10^{-3}$ im Mittel also jedes zehnte falsch – das ist nicht besonders gut.

Im letzten Schritt werden die erkennbaren und nicht erkennbaren Zweibitfehler getrennt, da diese Fälle noch überschaubar sind. Zweibitfehler bleiben unerkannt, wenn die Verfälschungen in gleichwertigen Bitstellen auftreten. Mit acht Datenbytes und einem Prüfbyte existieren pro Bitstelle $8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 = 9 \cdot 8/2 = 36$ unterschiedliche Fehlermöglichkeiten[36]. Daraus folgen mit acht Bits pro Byte insgesamt $8 \cdot 36 = 288$ Fehlermöglichkeiten. Gut die Hälfte kann nicht erkannt werden. Bei den Datenbytes müssen die Verfälschungen invers sein, damit die Prüfsumme konstant ist (Subtraktion). Wenn die Prüfsumme mitverfälscht wird, müssen die Änderungen hingegen identisch sein (gleiche Operation links und rechts vom Gleichheitszeichen). Bei gleichhäufigen „0-1"- und „1-0"-Bitfehlern ergibt das genau die Hälfte, da die „0en" und „1en" in Codewörtern im Mittel genauso oft vertreten sind. Etwas mehr als die Hälfte kommt aufgrund des endlichen Zahlenkörpers zustande. In den MSBs der Datenbytes bleiben zusätzlich die Fälle unerkannt, bei denen zwei Bits auf dieselbe Weise verfälscht werden ($(\pm 128 \pm 128) \bmod 256 = 0$). Deshalb wird die Anzahl der nicht erkennbaren Zweibitfehler auf 150 geschätzt. Somit ergibt sich für die Restfehlerwahrscheinlichkeit des „Resonse"-Teils im LIN-Telegramm:

$$P_{Restfehler} \approx 1 - P_{(Restfehler|0\ Bitfehler)} - P_{(Restfehler|1\ Bitfehler)} - (2556 - 150) \cdot P_{Bitfehler}^2 \cdot (1 - P_{Bitfehler})^{70}$$

$$P_{Restfehler} \approx 1{,}96 \cdot 10^{-4} \qquad \text{für } P_{Bitfehler} = 10^{-3}$$

Formel 2.4: Restfehlerwahrscheinlichkeit bei einem acht Datenbyte langem LIN-Telegramm

Mit dieser Restfehlerwahrscheinlichkeit erreicht der LIN-Bus, selbst mit periodischer Wiederholung der Daten, keine Datenintegritätsklasse nach DIN 19244 (siehe Punkt 8.7). Eine Paritybit-Sicherung[37] für jedes Datenbyte wäre, bei gleichem Aufwand, um den Faktor 5 besser, wodurch man die Datenintegritätsklasse I1 erreichen könnte. Angesichts dieses Ergebnisses ist die Fehlersicherung mit der Modulo-Summe fragwürdig. Das gilt für die Invertierung ebenso, sie macht Aufwand ohne erkennbaren Nutzen.

Bei kleineren Datenmengen wird die Restfehlerwahrscheinlichkeit jedoch etwas geringer. Denn weniger Datenbytes bieten weniger Möglichkeiten für Bitverfälschungen. Prinzipiell wäre das Minimum bei einem Datenbyte erreicht. Das kann zwar beim LIN-Bus nicht vorkommen (min. zwei Datenbytes), doch gestattet dieser Fall eine rasche Abschätzung des „Best-Cases". Mit einem Datenbyte ist die Prüfsumme nur eine Kopie, unabhängig von der Invertierung. Deshalb müssen bei der Berechnung der Restfehlerwahrscheinlichkeit Fehlerpaare berücksichtigt werden. Im nächsten Abschnitt wird genau dieser Fall beim PPC-Bus behandelt. Nach Anpassung der Formel 2.5 ($e_{max} = 8$, $e = 1..8$) erhält man, für eine mittlere Bitfehlerwahrscheinlichkeit von 10^{-3}, eine Restfehlerwahrscheinlichkeit von $\approx 7{,}9 \cdot 10^{-6}$. Aufgrund der Restriktion von min. zwei Datenbytes, wird das Minimum der Restfehlerwahrscheinlichkeit auf 10^{-5} geschätzt. Damit könnte ein zyklisch betriebener LIN-Bus die Datenintegritätsklasse I1 erreichen.

Im „Header"-Teil befinden sich nur im „Ident"-Byte Daten. Nach Bild 2.7 enthält das „Ident"-Byte die Paritybits P0 und P1 und die sechs Informationsbits ID0 bis ID5. Die Paritybits werden nach den folgenden Vorschriften berechnet:

---

[36] Bei einer bestimmten Wertigkeit hat das erste Bit acht Kombinationsmöglichkeiten, in den restlichen acht Bytes. Das nächste Bit hat aber nur noch sieben, weil sonst ein Fall doppelt auftritt usw.

[37] Mit einer Parity-Sicherung und acht Datenbits erreicht man für $P_{Bitfehler} = 10^{-3}$ eine Restfehlerwahrsch. von $\approx 3{,}57 \cdot 10^{-5}$.

$$P0 = ID0 \oplus ID1 \oplus ID2 \oplus ID4 \qquad GF(2)$$

$$\overline{P1} = ID1 \oplus ID3 \oplus ID4 \oplus ID5 \qquad GF(2)$$

Das Paritybit P0 entspricht gerader Parität und das Paritybit P1 ungerader Parität. Normalerweise müsste man für die Restfehlerwahrscheinlichkeit die Verkopplung durch die Bits ID1 und ID4 beachten. Allerdings genügt hier eine Abschätzung, wodurch eine getrennte Betrachtung beider Teile mit der Formel:

$$P_{Restfehler} < \sum_{e} \binom{5}{e} \cdot P_{Bitfehler}^{e} \cdot (1 - P_{Bitfehler})^{5-e} \qquad \text{für } e = 2,\ 4$$

erlaubt ist. Sie liefert für ebenfalls $P_{Bitfehler} = 10^{-3}$ eine Restfehlerwahrscheinlichkeit, die etwas kleiner als 10^{-5} ist. Kleiner deshalb, weil durch die Verschränkung von ID1 und ID4 ein paar Fehlerfälle mehr erkannt werden als angenommen. Wegen der schlechteren Restfehlerwahrscheinlichkeit des „Response"-Teils, spielt der „Header"-Teil in der Gesamtrestfehlerwahrscheinlichkeit allerdings keine Rolle.

2.3.4 PPC-Bus

Allgemeines

Der PPC-Bus ist ein weiteres System für die Steuerung von Komfortelektronik und nicht sicherheitsrelevante Einrichtungen in Kraftfahrzeugen. Entwickelt wurde bzw. wird dieser serielle „low-cost" Sensor/Aktorbus von der niederländischen Firma Power-Packer und dem bekannten Tastaturhersteller Cherry. Letzterer hat den PPC-Bus auf dem ZVEI-Podium der Messe electronica 2000 erstmals vorgestellt. Eine Spezifikation ist noch nicht zugänglich. In zwei Aufsätze wird beschrieben, dass der PPC-Bus eine Konkurrenz zum LIN-Bus bilden soll [60] [61]. Power-Packer und Cherry bezeichnen den PPC-Bus als so genanntes „Niche-Area-Network" (NAN). Mit einem ASIC und hohen Stückzahlen soll er dem Kostendruck von ungefähr 0,5..1 € pro Knoten standhalten [71].

Funktionsprinzip

Als Arbitrierungsprinzip kommt das „Master/Slave"-Verfahren zum Einsatz. Im Adressraum ist Platz für 30 „Slaves", die im System programmierbar sind. Das Baudratenspektrum deckt den Bereich 10..32 kBit/s ab. Für die Kommunikation wurde ein asynchrones, UART-kompatibles Verfahren ausgewählt. Der „Physical-Layer" ist an die Norm ISO 9141 angelehnt. Sämtliche Teilnehmer werden mittels „Wired-AND"-Technik an einen ungeschirmten Draht angeschlossen. Als max. Netzausdehnung sind 10 m spezifiziert. Das ist verhältnismäßig wenig, reicht jedoch für viele Kfz-Baugruppen aus (Tür, Sitz usw.). Aufgrund der kapazitiven Last bei einem vollausgebauten PPC-Bus, ist die max. „Slave"-Anzahl in der Praxis auf 20 reduziert. Eine Besonderheit des PPC-Busses ist die Synchronisation mit PLL-Oszillatoren. Hiermit können alle „Slaves" die „Master"-Baudrate, über einen nicht näher beschriebenen Mechanismus, sehr genau einstellen. Für einen „Slave" benötigt man nur den so genannten PPC-ASIC. Als „Master" genügen wahrscheinlich µCs mit UART-Schnittstelle.

Telegramm

Daten werden zwischen dem „Master" und einem „Slave" zeichenweise in 9N1-UART-„Frames" ausgetauscht. Diese elf Bitzeiten langen „Frames" besitzen ein Startbit, neun Datenbits und ein Stoppbit. Anstelle der gängigen Parity-Sicherung, arbeitet der PPC-Bus mit serieller Redundanz. Alle „Frames"

werden zweimal gesendet. Mehr ist über die Telegramme zur Zeit leider nicht bekannt. Da es sich um einen „Master/Slave“-Bus handelt, muss ein Telegramm jedoch aus einem „Master“-Aufruf und einer „Slave“-Antwort bestehen. Es wird angenommen, dass jeder Teil zwei „Frames“ enthält (inkl. Wiederholung). Zudem dürfte ein Telegramm mit einem Synchronisations-„Frame“ beginnen.

Zeitverhalten

Je nach Baudrate dauert ein PPC-„Frame“ $11 \cdot (31{,}25..100\ \mu s) \approx 344\ \mu s..1{,}1$ ms. Unter der Annahme des beschriebenen Telegramms plus 10 % für Pausen, wird die Telegrammdauer auf 2..6 ms geschätzt[38]. Bei einer vernünftigen Anzahl von „Slaves“ (≈ 20), könnten Abtastperioden größer 50 ms erreichbar sein. Die Realzeitfähigkeit des PPC-Busses wird als bedingt eingeschätzt. Denn erstens ist sie für einfache Kfz-Anwendungen nicht nötig. Und zweitens ist eine Zeitsteuerung, so wie bei den meisten anderen Bussen, Sache der Applikation. Deshalb muss auch mit einem größeren „Jitter“ gerechnet werden. Die beiden Firmen hätten sicher ausdrücklich darauf hingewiesen, wenn dem nicht so wäre. Teilweise definiert ist hingegen das Zeitverhalten im Fehlerfall. Ein „Bus-Timeout“ von 250 ms ermöglicht den „Slaves“ Unterbrechungen zu erkennen.

Effizienz

Die „Frame“-Effizienz ist mit 9 Bit/11 Bit $\approx 0{,}82$ hoch. Aufgrund der Wiederholungen, der angenommenen Pausenzeiten, der Synchronisations-„Frames“ und der „Master/Slave“-Arbitrierung, unterscheidet sich die Gesamteffizienz davon stark. Den oberen Ausführungen nach ist ein Telegramm ca. 61 Bitzeiten lang. Somit reduziert sich die Gesamteffizienz auf $9 \cdot T_{Bit} / 61 \cdot T_{Bit} \approx 0{,}15$. Aufgrund der Ungewissheiten wird für sie ein Bereich von 0,1..0,2 geschätzt.

Verlässlichkeit

Einen Vorteil besitzt der PPC-Bus, weil der „Master“ alle „Frames“ mithört. Das ist nicht selbstverständlich wie das Beispiel ASI-Bus zeigt. Durch das Mithören soll der PPC-„Master“ vor allem Leitungsdefekte erkennen. Für die Sicherung gegen Übertragungsfehler werden die „Frames“ zweimal übertragen, wobei die Datenbits im zweiten „Frames“ invertiert sind. Ein erkannter Fehler führt zur Verwerfung des Telegramms. Leider ist diese Art der Fehlersicherung nicht sehr effizient. Die Hamming-Distanz d_{min} ist zwei und damit nicht besser als mit einem Paritybit. Außerdem verbessert die Invertierung der Datenbits die Fehlererkennungseigenschaften nicht. Sie erzeugt nur Rechenarbeit. Bei paralleler Redundanz ist das anders, weil dort ein Störimpuls sehr selten zwei gleichzeitig verschickte, verschiedene Bits kippt. Eine Invertierung würde mit serieller Redundanz die Fehlererkennung nur verbessern, wenn bei den Bitverfälschungen eine Vorzugsrichtung existiert. In einem BSC-Kanal (siehe Abschnitt 8.2), mit oder ohne „Burst“-Fehler, der eine drahtgebundene Übertragung gut widerspiegelt, trifft das jedoch nicht zu.

Gegenüber einer Parity-Sicherung hat die Wiederholungsmethode, trotz gleicher Hamming-Distanz, eine etwas bessere Restfehlerwahrscheinlichkeit. Mit einem Paritybit werden alle 2-, 4-, 6- und 8-Bitfehler nicht erkannt. Bei einer Wiederholung trifft das nicht zu, weil die ebenfalls geradzahligen Bitfehler zusätzliche Bedingungen erfüllen müssen. Bitfehler bleiben nur dann unerkannt, wenn sie in beiden Bytes an der gleichen Stelle auftreten. Diese Einschränkung reduziert die Anzahl der Kombinationen, wodurch die Restfehlerwahrscheinlichkeit geringer ausfällt. Für ihre Berechnung kann man eine angepasste Binomialverteilung verwenden:

[38] Es wird folgende Formel angenommen: $T_{Telegramm} = 11 \cdot T_{Bit} \cdot 5 \cdot 1{,}1$ (5: „Frames“ pro Telegramm; 1,1: Pausen).

$$P_{Restfehler} = \sum_{e} \binom{9}{e} \cdot P_{Bitfehler}^{2 \cdot e} \cdot (1 - P_{Bitfehler})^{2 \cdot (9-e)} \quad \text{für} \quad e = 1 \ldots 9$$

Formel 2.5: Restfehlerwahrscheinlichkeit beim PPC-Bus

Im Binomialkoeffizient geht man von einem Datum aus, das ab einem Bitfehler unerkannt gestört werden kann. Durch die Paarbildung, die sich in den Exponenten auswirkt, entstehen geradzahlige Bitfehler. Entsprechend zählt man mit der Laufvariablen e Bitpaare, anstelle von einzelnen Bits.

Für die durchwegs angenommene Bitfehlerwahrscheinlichkeit von 10^{-3}, errechnet sich aus Formel 2.5 eine Restfehlerwahrscheinlichkeit von ca. $8{,}86 \cdot 10^{-6}$. Mit einem Paritybit und neun Datenbits wären es ca. $3{,}57 \cdot 10^{-5}$ (Faktor vier). Wegen der höheren Bitanzahl, ist bei der Wiederholmethode die Wahrscheinlichkeit für ein korrekt übertragenes Datum allerdings geringer. Sie beträgt $(1\text{-}P_{Bitfehler})^{18} = 0{,}98$. Mit einer entsprechenden Parity-Sicherung wären es $(1 - P_{Bitfehler})^{10} = 0{,}99$ (jeweils für $P_{Bitfehler} = 10^{-3}$). Sofern man den PPC-Bus zyklisch betreibt, erfüllt er, im Gegensatz zum LIN-Bus, die Kriterien der Datenintegritätsklasse I1 nach DIN 19244 (siehe Abschnitt 8.7).

2.3.5 BST-Bus

Allgemeines

Die Firmen Bosch, Siemens und Temic haben für Unfallschutz- und Sicherheitseinrichtungen in zukünftigen Kfzs einen so genannten Zündbus entwickelt. Der Ausdruck Zündbus rührt von den pyrotechnischen Aktoren her. Auf ihnen befindet sich je eine Sprengkapsel, die durch Explosion die, in sehr kurzer Zeit, benötigte Energie zum Aufblasen von „Airbags“ oder Straffen von Gurten liefert. Bis heute steuert man diese Unfallschutz- und Sicherheitseinrichtungen separat, mit relativ leistungsstarken µCs und eigenen Sensoren. Im Falle einer Zündung werden diese Bauteile zerstört. Durch die wachsende Anzahl der Unfallschutz- und Sicherheitseinrichtungen ist diese Technik zu teuer geworden. Vor ein paar Jahren verbaute man für Fahrer und Beifahrer zwei „Airbags“ und zwei Gurtstraffer. Heute sind es bereits sechs bis acht dieser Einrichtungen und in der Zukunft rechnet man mit mehr als 20. Deshalb, und aus Verlässlichkeitsgründen, sollen diese Einrichtungen über sichere Sensor/Aktorbusse, wie dem BST-Bus, gesteuert werden. Als Kostenspanne pro Knoten sind 1..2 € vorgesehen [71].

Eine komplette Neuentwicklung ist der BST-Bus nicht. Er entstand aus dem BoTe- (Bosch, Temic) und SURFS-Bus (Siemens: Standard Universal Remote Firing System) aufgrund einer Kooperation. Seine aktuelle Spezifikation trägt die Version 2.00 und ist z.B. auf den „Web“-Seiten von Temic zugänglich [67]. Dort wird der BST-Bus unter der Bezeichnung „Restraint-System-Bus“ (RSB) bereits beworben. Verfügbar wird er jedoch erst in ein paar Jahren sein. Laut SiemensVDO sind ASICs für 2004 geplant und nach Aussagen von Bosch erfolgt gerade eine ISO-Standardisierung.

Funktionsprinzip

Der serielle BST-Bus arbeitet nach dem klassischen „Master/Slave“-Prinzip. Ein Busstrang verfügt über einen „Master“ und bis zu 62 „Slaves“. Zwölf von ihnen, die so genannten „Squib-Slaves“, können pyrotechnische Aktoren besitzen[39]. Der Rest sind gewöhnliche „Slaves“, z.B. für „Crash“-Sensoren. Die Baudraten sind 31,25 kBit/s, 125 kBit/s und 250 kBit/s. In der Regel werden die Zünddaten mit den höheren Baudraten übertragen und z.B. Diagnosedaten mit den niedrigeren. Als Topologie

[39] squib (engl.) = Knallfrosch

wird eine Linienform mit einer Ausdehnung von bis zu 30 m verwendet. Beim Anschluss eines „Slaves“ muss man auf den Typ achten. Es gibt Parallel- oder „Daisy-Chain-Slaves“; der „Master“ ist immer ein Paralleltyp. Ein Busstrang darf parallel-, „Daisy-Chain“- oder gemischt verdrahtet sein. Aus Gründen der Störsicherheit (EMV, „Stuck-at-Fehler“) findet die Kommunikation differentiell statt, wobei Energie und Daten über das gleiche Leitungspaar übertragen werden. Hierfür verändert der „Master“, in Abhängigkeit von den Daten, die Busspannung. Hohe Spannungen (12 V) dienen dem Energietransport und die Spannungswechsel der Kommunikation. Eine richtige „Powerline“, so wie z.B. beim ASI-Bus, ist das nicht, da der „Master“ die Energiequelle für die „Slaves“ ist. Durch die Verlustleistung im Bustreiber, ist der dauerhafte Ausgangsstrom des „Masters“ auf ca. 170 mA begrenzt. Je nach Strombedarf der „Slaves“, kann das ihre max. Anzahl pro Busstrang deutlich reduzieren.

Für die Datenübertragung nutzt der BST-Bus ein synchrones Vollduplex-Verfahren. Der „Master“ sendet seine Daten, wie beschrieben, spannungsmoduliert. Die „Slaves“ dagegen antworten strommoduliert, wodurch zwei Kanäle entstehen. Im Gegensatz zum DSI-Bus, nutzt der BST-Bus die Vollduplex-Fähigkeit nicht zur Erhöhung der Übertragungseffizienz. Er synchronisiert damit lediglich die „Slaves“. Die Synchronisation geschieht primär über eine Manchester-Codierung, mit der eine Kommunikation trotz großer Oszillatortoleranzen möglich ist. Aus diesem Grund kommen die „Slaves“ mit einem kostengünstig Oszillator aus. Ohne weiteres zutun würden die „Slaves“-Antworten zeitlich stark schwanken. Um das zu verhindern, versendet der „Master“ Leertelegramme während die „Slaves“ antworten. Aus den Leertelegrammen extrahieren die „Slaves“ den „Master“-Takt und steuern damit zeitgenau ihre Datenübertragung.

Zu den Besonderheiten des BST-Busses zählt, neben der quasi „Powerline“, die Priorisierung der so genannten „Deploy-Frames“ für die Zündung der „Squib-Slaves“. Im Gegensatz zu „Normal-Frames“, die Diagnose- und Konfigurationsdaten beinhalten, werden „Deploy-Frames“ wesentlich stärker moduliert. Die Differenzspannung beträgt bei „Normal-Frames“ etwa 5 V und bei „Deploy-Frames“ etwa 12 V. Dadurch können die „Slaves“ ein „Deploy-Frame“ sofort erkennen, wodurch die Unterbrechung von „Normal-Frames“, so wie in Bild 2.8 gezeigt, möglich ist[40]. Für eine Zündung ist das von Vorteil, weil keine zusätzlichen Wartezeiten entstehen. Außerdem macht die höhere Differenzspannung „Deploy-Frames“ störunempfindlicher als „Normal-Frames“. Und schließlich verhindert sie, dass nur ein paar Bitverfälschungen ein „Normal-Frame“ in ein „Deploy-Frame“ verwandeln können.

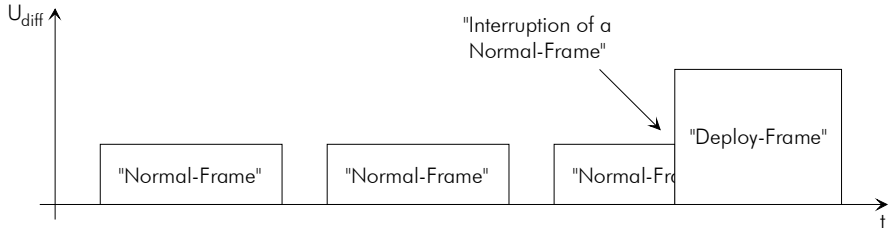


Bild 2.8: Telegrammunterbrechung beim BST-Bus

Telegramme

Der BST-Bus unterscheidet vier Telegramme: Lesen von Daten aus einem „Slave“ („Read-Frame“), Schreiben von Daten in einen „Slave“ („Write-Frame“), Lesen von Daten aus einer „Slave“-Gruppe („Polling-Frame“) und Zünden von „Squib-Slaves“ („Deploy-Frame“). Alle Telegramme sind ähnlich

[40] Damit gehört der BST-Bus zu den bitorientierten Bussen.

aufgebaut und es gilt durchwegs die Regel „MSB first“. „Read-“ und „Write-Frames“ sind fast identisch, wie Bild 2.9 zeigt. Beide Telegramme sind 42 Bits lang und beginnen mit einer logischen Doppelnull als „Start Delimiter“. Darauf folgt ein Dreibitbefehl („Command“) mit ungerader Paritybit-Sicherung. Für ein „Read-Frame“ lautet diese Bitfolge „1000“ und für ein „Write-Frame“ „1110“. Entsprechend Bild 2.9, wird im Anschluss eine Sechsbitadresse ($A_5..A_0$) zur Auswahl eines „Slaves“ verschickt. Von den $2^6 = 64$ Adressen sind zwei Adressen reserviert: Adresse 0 spricht unprogrammierte „Slaves“ an und Adresse 63 alle „Slaves“ („Broadcast Command“). Somit verbleiben für die individuelle „Slave“-Adressierung die Nummern 1..62.

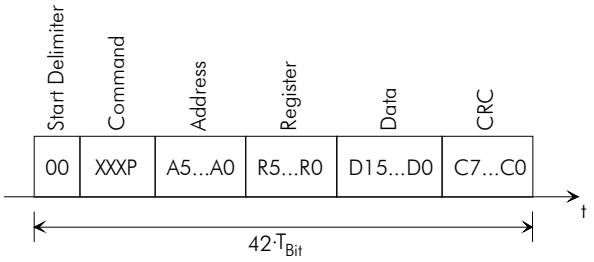


Bild 2.9: „Read-„ und „Write-Frame“ beim BST-Bus

Mit den Bits $R_5..R_0$ wird ein „Slave“-Register ausgewählt. Bei einem „Write-Frame“ werden die Datenbits D15..D0 dort abgelegt, sofern die CRC-Prüfung keine Fehler anzeigt. Im Falle eines „Read-Frames“ sendet der „Master“ in D15..D0 und C7..C0 logische „1en“ für die Taktung des angesprochenen „Slaves“. Dieser schickt ihm den gewünschten Registerinhalt und das CRC-Prüfwort auf dem Stromkanal parallel zurück. Mit dem CRC-Prüfwort werden alle Bits nach dem „Start Delimiter“ gesichert. Das Generatorpolynom hierfür und für alle weiteren Fehlersicherungen lautet: $g_8(x) = x^8 + x^7 + x^6 + x^4 + x^2 + 1$.

Das Format von „Polling-Frames“ ist in Bild 2.10 dargestellt. Mit einem „Polling-Frame“ kann der „Master“ von mehreren „Slaves“ zugleich lesen. Dazu muss man vorher Gruppen definieren (max. 16), die im Betrieb mittels der (Gruppen)Adressbits $G_3..G_0$ angesprochen werden. Für die Datenübertragung gilt das Gleiche wie für „Read-Frames“: Der „Master“ sendet in den Bits DN..D0 logische „1en“, währenddessen die „Slaves“ nacheinander bestimmte Registerwerte senden. Aufgrund unterschiedlicher Gruppengrößen, ist die Länge des Datenfeldes nicht konstant. Je nach Festlegung, kann sie zwischen 8..48 Bits betragen. Dadurch variiert die Länge in einem Bereich von 18..58 Bits. Gegen Übertragungsfehler sind „Polling-Frame“ leider nicht gesichert.

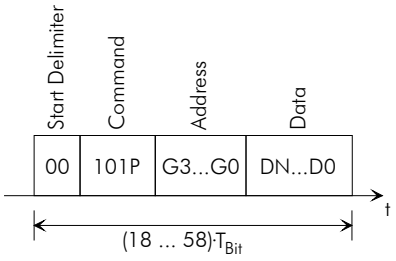


Bild 2.10: „Polling-Frame“ beim BST-Bus

Das letzte Telegramm ist das 26 Bit lange „Deploy-Frame“ (siehe Bild 2.11). Nach dem standardmäßigen „Start Delimiter“ und „Command“-Teil, folgen die so genannten „Deploy Flags“ (D11..D0). Jedes

dieser zwölf Bits ist für die Zündung eines „Squib-Slave“ verantwortlich. Damit sie gleichzeitig ansprechbar sind, handelt es sich hier um einen „Multicast“. Mit dem abschließenden CRC-Prüfbyte wird hauptsächlich die Wahrscheinlichkeit von Falschzündungen minimiert (z.B. Zündung des Beifahrer- anstelle des Fahrer-„Airbags“). Für die Sicherung gegen Fehlzündungen (Zündung ohne Unfall) spielen vor allem die separaten „Deploy-Frame“-Symbole und die Telegrammlänge eine Rolle.

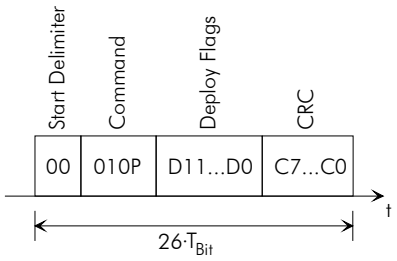


Bild 2.11: „Deploy-Frame“ beim BST-Bus

Zeitverhalten

Durch die Zweikanaligkeit, in Kombination mit der synchronen Übertragung, gibt der „Master“ stets den Bittakt vor. Aus diesem Grund ist der BST-Bus praktisch „Jitter“-frei. Zwischen den Telegrammen fügt der „Master“ eine mindestens vier Bitzeiten lange Pause ein (siehe Bild 2.12); einzige Ausnahme ist die Unterbrechung durch ein „Deploy-Frame“. Auf ein Schreibtelegramm („Write-“ oder „Deploy-Frame“) reagiert ein „Slave“ nach der dritten Pausenbitzeit. Er macht das als zusätzliche Fehlersicherung. Falls eine Pause kleiner als drei Bitzeiten ist, so verwirft ein „Slave“ das Telegramm zuvor und danach, weil beide Telegramme diesen Fehler verursachen können.

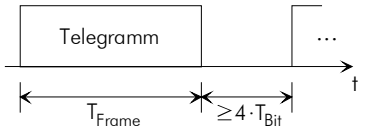


Bild 2.12: Standardzeitverhalten des BST-Busses

In der Regel wird der BST-Bus zyklisch arbeiten, damit der „Master“ stets die Beschleunigungswerte der „Crash“-Sensoren kennt – leider muss man das explizit programmieren. Aufgrund des Sicherheitsaspekts eignen sich „Polling-Frames“ hierfür weniger. Sie wären zwar effizient, besitzen aber keine Fehlersicherung. Deshalb werden für eine Zeitbetrachtung nur die gesicherten, 42 Bits langen „Read-Frames“ berücksichtigt. Gemäß Bild 2.12, kann der „Master“ damit alle $(42 + 4)\cdot T_{Bit}$ ein neues Telegramm absetzen. Bei einer Baudrate von 31,25 kBit/s entspricht dieser Zeitraum 1,47 ms, bei 125 kBit/s 0,368 ms und bei 250 kBit/s 0,184 ms. Stellt man diesen Zeiten die kritischste Verarbeitungszeit von 3 ms (Seitenaufprall) gegenüber [23], so wird klar, dass nur die beiden Baudarten 125 kBit/s und 250 kBit/s für die Zündung von „Airbags“ usw. geeignet sind.

Insbesondere gilt das für die Baudrate 250 kBit/s, wie folgendes Beispiel zeigt: Angenommen werden sechs „Slaves“ mit „Crash-Sensoren“ (je zwei vorne, links und rechts). Der „Master“ soll sie periodisch auslesen, mittels gesicherter „Read-Frames“. Bei einer Baudrate von 250 kBit/s und minimaler Pausendauer ergibt das eine Abtastzeit von $6\cdot(42 + 4)$ Bit/250 kBit/s ≈ 1,1 ms. Für das Zünden der „Airbags“ durch ein „Deploy-Frame“ werden (26 + 3)/250 kBit/s ≈ 0,1 ms benötigt. Etwas später wird

gezeigt, dass der BST-Bus für eine sichere Zündung ein „Deploy-Frame“ mindestens dreimal wiederholen sollte. Daraus folgt eine max. Übertragungszeit von ca. 1,4 ms. Unter Berücksichtigung der mittleren, abtastbedingten Totzeit von 0,55 ms, vergehen im Ernstfall somit knapp 2 ms. Bei einem kritischen Seitenaufprall dürfen die Berechnungen also gut 1 ms dauern. Für größere µCs oder DSPs ist das kein Problem. Schwierig wird das Ganze bei einer Baudrate von 125 kBit/s. Dort würden sich die Übertragungszeiten und Abtasttotzeit auf etwa 4 ms summieren, weshalb man mit ungesicherten „Polling-Frames“ arbeiten müsste.

Effizienz

Bei der Telegrammeffizienz schneidet der BST-Bus relativ gut ab. Ein „Read-“ oder „Write-Frame“ kommt auf 16 Bit/42 Bit ≈ 0,38 und ein „Deploy-Frame“ auf 12 Bit/26 Bit ≈ 0,46. „Polling-Frames“ werden nicht berücksichtigt, da sie keine Fehlersicherung besitzen. Die Gesamteffizienz kann nur abgeschätzt werden. Aufgrund der expliziten Zeitsteuerung in der „Master“-Applikation, können die Telegrammpausen nämlich beliebig lang sein. Das ist jedoch genauso wenig praxisrelevant wie die min. Telegrammpause mit vier Bitzeiten. Bei den höheren Baudraten beträgt diese lediglich 16 bzw. 32 µs. Einige µCs oder DSPs sind dafür schnell genug, andere hingegen nicht. Außerdem spielen die Applikation und „Critical-Sections“ eine unbekannte Rolle. Deshalb wird von einer durchschnittlich zehn Bitzeiten langen Pause ausgegangen. Weiterhin werden nur „Read-“ und „Write-Frames“ zugelassen. Mit diesen Annahmen beträgt die Gesamteffizienz 16 Bit/52 Bit ≈ 0,31. 16 Datenbits sind für Sensor/Aktoranwendungen relativ viel. Auch in diesem speziellen Anwendungsfall, in dem hauptsächlich Beschleunigungen gemessen werden. Aufgrund des Rauschens ist eine Auflösung von 8..12 Bit wahrscheinlicher als eine 16 Bit Auflösung, so dass die Gesamteffizienz eher im Bereich 15..25 % liegt.

Verlässlichkeit

Ein unmotiviert platzender „Airbag“ kann während der Fahrt einen Unfall verursachen. Neben den Sachschäden und der Rufschädigung des Herstellers, besteht die Gefahr der Tötung von Insassen und anderen Personen als Folge des Unfalls. Deshalb sehen die Automobilfirmen diese so genannten Fehlzündungen am kritischsten an. Alle anderen Fehlerfälle (z.B. Verfälschung eines „Read-Frames“ oder Zündung eines falschen „Airbags“) sind zwar nicht unwichtig, spielen bei weitem jedoch keine so große Rolle. Deswegen wird in diesem Abschnitt nur auf die Wahrscheinlichkeit von Fehlzündungen eingegangen. Die Berechnungen sind allerdings so komplex und aufwändig, dass es keinen Sinn macht sie hier detailliert aufzuzeigen. Man kann sie, und andere Wahrscheinlichkeiten des BST-Busses, in der Studie [69] nachlesen. Stattdessen sollen die wichtigsten Zusammenhänge und Ergebnisse dargelegt werden.

Als erstes gilt es, die Voraussetzungen für eine Fehlzündung zu klären. Die betroffenen „Squib-Slaves“ zünden, wenn sie ein gültiges „Deploy-Frame“ empfangen haben, gefolgt von einer drei Bitzeiten langen Pause. Weiterhin kommen „Deploy-Frames“ nur bei einem Unfall vor, weshalb eine Fehlzündung nur durch die Verfälschung eines „Normal-Frames“, einer entsprechend langen Pause oder aus der Kombination beider entstehen kann. Bekanntermaßen verwendet der BST-Bus für „Deploy-Frames“ andere Symbole als für „Normal-Frames“. Es ist deshalb nicht möglich, mit nur ein paar Bitfehlern aus einem „Normal-Frame“ ein „Deploy-Frame“ zu erzeugen. Man benötigt dazu eine ganze Reihe aufeinanderfolgender, verfälschter Symbole. Inklusiv der Pause sind es 29 Stück, die nur in wenigen Anordnungen ein gültiges Codewort ergeben. Da so viele Bedingungen erfüllt werden müssen, ist die Wahrscheinlichkeit für eine Fehlzündung extrem klein. Sie kann in erster Näherung mit der Formel 2.6 abgeschätzt werden.

$$P_{\text{Fehlzündung}} < 2^{-8} \cdot P_{\text{Bitfehler}}^{29}$$

Formel 2.6: Abschätzung der Fehlzündwahrscheinlichkeit beim BST-Bus

Der Exponent 29 berücksichtigt die Erzeugung der „Deploy-Frame"-Symbole und der Faktor 2^{-8}, dass dabei zufällig ein gültiges „Deploy-Frame" herauskommt. Bei einer mittleren Bitfehlerwahrscheinlichkeit von 10^{-3} liefert Formel 2.6 eine Fehlzündwahrscheinlichkeit kleiner 10^{-90}. Dieser Wert liegt ein Stück über dem Ergebnis aus der erwähnten Studie, wie der Graph in Bild 2.13 zeigt. Allerdings spielen die ca. 20 Zehnerpotenzen Unterschied bei diesem Absolutwert keine Rolle mehr. Was diese Zahl für den Kfz-Alltag bedeutet, soll ein Beispiel aus der Studie zeigen: Ein Kfz wird pro Jahr durchschnittlich ca. 416 Stunden lang betrieben. Bei einer Baudrate von 125 kBit/s ergibt das 125 kBit/s·416 h·3600 s/h ≈ $187{,}2 \cdot 10^{11}$ Bits bzw. Symbole pro Jahr und Kfz. Aus diesen Symbolen entstehen, bei einer mittleren Bitfehlerwahrscheinlichkeit von 10^{-4}, max. $187{,}2 \cdot 10^{11} \cdot 10^{-4}/26 = 720 \cdot 10^{3}$ „Burst"-Fehler der Länge 26. Von diesen „Burst"-Fehlern werden im Mittel $720 \cdot 10^{3} \cdot 10^{-140} = 7{,}2 \cdot 10^{-135}$ pro Jahr und Auto eine Fehlzündung bewirken. Im Durchschnitt vergehen in einem Auto also gut 10^{134} Jahre zwischen zwei Fehlzündungen. Bezogen auf die ca. 500 Millionen Kfzs weltweit, passiert das dann alle 10^{129} Jahre. Trotz dieses astronomischen Wertes, darf man sich nicht in Sicherheit wiegen. Denn praktisch gesehen sagt die niedrige Fehlzündwahrscheinlichkeit nur aus, dass andere Systemkomponenten viel eher eine Fehlzündung bewirken als der Informationskanal.

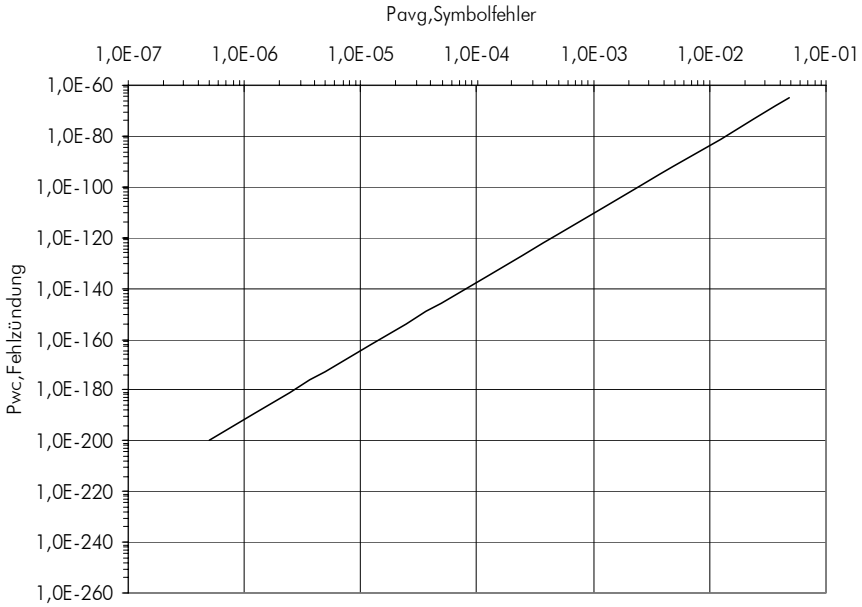


Bild 2.13: Fehlzündwahrscheinlichkeit des BST-Busses [69]

Das Ganze hat jedoch auch eine Kehrseite. In einem echten „Deploy-Frame", inklusiv Pause, reicht bereits ein Fehler aus, damit z.B. ein „Airbag" nicht zündet. Aufgrund der Telegrammlänge gibt es sehr viele Fehlermöglichkeiten, weshalb die Nichtzündwahrscheinlichkeit relativ gering ausfällt. Bei einer mittleren Bitfehlerwahrscheinlichkeit von 10^{-3} beträgt sie etwa 10^{-2}. Im Durchschnitt würde also jeder hunderste „Airbag" bei einem Unfall nicht zünden. Deshalb muss der BST-Bus mit mindestens drei Wiederholungen arbeiten, um eine Nichtzündwahrscheinlichkeit von 10^{-6} zu erreichen.

2.3.6 CAN-Bus

Allgemeines

Das „Controller-Area-Network“ (CAN) wurde von 1983 bis 1986 von den Firmen Bosch und Intel für Automobile entwickelt. Ein Jahr später bot Intel, mit dem 82526, den ersten CAN-„Controller“ an. Kurz danach folgte der 82C200 von Philips. Seinen Durchbruch verdankt der CAN-Bus jedoch der Automatisierungstechnik. Er fand dort relativ schnell und in vielen Bereichen Anwendung. Aus diesem Umfeld entstand 1992 die Nutzerorganisation „CAN in Automation“ (CiA). Mit der zunehmenden Bedeutung wuchs die Anzahl der CAN-Bausteine. Um 1996 gab es bereits mehr als 15 Stück Bausteine [39]. Außerdem waren die Stückzahlen schon so hoch, dass die Preise sanken. In dieser Zeit kostete ein CAN-„Controller“ ca. 5 € und ein SLIO-Baustein[41] ca. 3 € [39]. Später zogen auch die Automobilhersteller nach. Mit dem Serieneinsatz des CAN-Busses in der S-Klasse, gehörte damals Daimler-Benz zu den Ersten. Weiterhin fand der CAN-Bus zunehmend in vielen anderen Bereichen (Medizintechnik, Antriebstechnik...) Anwendung, so dass im Jahr 1998 bereits über 20 Millionen CAN-Busstationen existierten [50]. Die Kosten-Stückzahlen-Spirale machte den CAN-Bus immer attraktiver. Nur zwei Jahre später wurden ca. 100 Millionen Bausteine in einem Jahr verkauft [16]. Heute zählt der CAN-Bus zu den bedeutensten Feldbussen, weshalb fast jeder Halbleiterhersteller mehrere CAN-Bausteine anbietet [31]. Dementsprechend günstig haben sich die Preise entwickelt. Integrierte „CAN/Host-Controller“ kosten teilweise weniger als die ehemaligen SLIO-Bausteine. Aus diesem Grund werden Letztere nicht mehr gebaut. Ein Sensor/Aktorbus war und ist der CAN-Bus aber ohnehin nicht. Dafür ist er zu aufwändig und immer noch zu teuer. Seine Domäne ist und bleibt die Geräteebene („Board“-Computer, Motronik, Servoantrieb, SPS...). Der CAN-Bus wird im Rahmen dieser Arbeit behandelt, weil er einer der wenigen ereignisgesteuerten Bussysteme ist.

In der CAN-Spezifikation sind der „Data-Link-Layer“ und „Physical-Layer“ beschrieben [27]. Die erste Norm erschien Ende 1993 unter der Bezeichnung ISO 11898. Im Laufe der Zeit wurde sie erweitert und aufgeteilt. Heute existieren für den CAN-Bus drei Normenteile: Den „Data-Link-Layer“ beschreibt ISO 11898-1, den „Physical-Layer“ ISO 11898-2 und einen „Fault-Tolerant-Physical-Layer“ ISO 11898-3. Mit ISO 11898-4 ist ein vierter Teil geplant bzw. in Arbeit [29]. Dort wird eine zeitgesteuerte Kommunikation für den CAN-Bus, namens „Time-Triggered-CAN“ (TTCAN), für sicherheitskritische Anwendungen definiert. Alternativ zum Standard-„Physical-Layer“ bis 1 MBit/s, kann man für „Low-Speed“-Applikationen bis 125 kBit/s den „Physical-Layer“ aus ISO 11519 verwenden. Erste Ansätze für eine höhere Protokollschicht wurden bereits vor 1990 in Schweden entwickelt. Sie mündeten in CAN-Kingdom [50], einer Art „Application-Layer“ mit einem „Real-Time-Operating-System for Systems“ (RTOS-S) [30]. Heute spielt CAN-Kingdom kaum noch eine Rolle. Der erste richtige „OSI Application-Layer“ entstand 1994 mit dem „CAN Application-Layer“ (CAL) von der CiA („Draft Standard 201 - 207“) [36]. CAL ist ein Defacto-Standard, konnte sich aber nie richtig durchsetzen, wegen eines fehlenden Anwenderprofils. Diese Lücke schloss CANopen, das CAL für die Definition eines Anwenderprofils für Produktionszellen nutzt. Nach einer Überarbeitung im Jahr 1995 wurde CANopen zu einem europäischen Standard (EN 50325). Heute hat CANopen eine große Bedeutung für die Vernetzung von „Embedded-Systems“ [50]. Des Weiteren enthält die Norm EN 50325 das DeviceNet. Es ist ein amerikanischer „Application-Layer“ für CAN, mit zusätzlichen US-spezifischen Definitionen im „Physical-Layer“ (Transceiver, Kabel, Stecker). DeviceNet spielt vor allem in der industriellen Automatisierung in Amerika eine Rolle.

[41] SLIO-Bausteine sind obsolet. Sie waren einfache CAN-Knoten (ohne „Host-Controller“) für den günstigen Anschluss von Sensoren und Aktoren, besaßen keine Intelligenz und unterstützten Baudraten bis 125 kBit/s.

Weitere „Application Layer“ bzw. Profile für CAN sind: „Smart-Distributed-System“ (SDS) von Honeywell, „Portable CAN Application Layer“ (PCAL) der Univerität der Bundeswehr München, der SAE-Standard J1939 für amerikanische LKWs und Omnibusse, das Landwirtschaftliche BUS-System (LBS) genormt in der DIN 9684 und das „Multiple Master Multiple Slave”-Anwenderprofil (M3S) für Rollstühle. Ihre Bedeutung ist jedoch eher gering. Außerdem wird in dem „Free-Software“-Projekt CANpie („CAN Programming Interface Environment“) versucht, eine Standard-API für CAN zu definieren. Initiator ist die Firma MicroControl, die das Projekt vor ca. 2 Jahren startete. Die API setzt auf den CAN-„Data-Link-Layer“ auf und soll u.a. Basis für höhere Protokollschichten werden, jedoch keine Konkurrenz dazu. Laut MicroControl verwenden bereits ein paar Firmen CANpie. Die aktuelle Version trägt die Nummer 0.6 [32]. Auf dem Sektor der sicheren Systeme stellt der CAN-Bus die Grundlage für den „SafetyBus“ der Firma Pilz, das Profil CANopen-Safety[42], das Profil DeviceNet-Safety und den zeitgesteuerten TTCAN-Bus dar.

Funktionsprinzip

CAN ist ein nachrichtenorientiertes Busprotokoll. Es arbeitet nicht mit Teilnehmeradressen, sondern mit Telegrammnummern, den so genannten „Identifiern“. Jeder „Identifier“ stellt eine Nachricht dar. Beim ursprünglichen „Standard-CAN“ (Spezifikation 2.0 A) beträgt die Anzahl der „Identifier“ 2032. Für den Einsatz in amerikanischen Lkws (US-Norm J1939), wurde diese Anzahl später durch die Spezifikation 2.0 B auf 536870912 erweitert. Zur Unterscheidung spricht man bei dieser Variante vom „Extended-CAN-Bus“. Ein „Extended-CAN-Bus“ kann „Standard-CAN“-Nachrichten verstehen, umgekehrt geht das aber nicht; neuere „Standard-CAN-Controller“ tolerieren „Extended-CAN“-Nachrichten, ältere reagieren darauf mit einer Fehlermeldung. Daneben existieren die beiden Begriffe „Basic-CAN“ und „Full-CAN“. Sie betreffen die „Standard-CAN-Controller“, haben aber nichts mit der Spezifikation 2.0 A zu tun. „Basic-CAN-Controller“ belasten den „Host-Controller“ relativ stark, weil sie jede Nachricht empfangen. Mit einer groben Gruppenfilterung kann man diese Belastung ein wenig verringern. Dagegen belasten „Full-CAN-Controller“ den „Host-Controller“ nur mit den nötigen Nachrichten. Sie besitzen einen so genannten Akzeptanzfilter, der bis zu 16 unterschiedliche Nachrichten selektiert und in „Mailboxes“ abgelegt. Beide „Standard-CAN-Controller“ können problemlos in einem Netz verwendet werden.

Als Arbitrierungsprinzip verwendet der CAN-Bus CSMA/CA („Carrier-Sense-Multiple-Access/Collision-Avoidance“). Bei dieser zerstörungsfreien Busarbitrierung warten die Teilnehmer mit einem Sendewunsch, bis der Bus frei ist (CS). Sie erkennen das an einer mindestens drei Bitzeiten langen Pause („Interframe-Space“) zwischen den Telegrammen. Erst dann beginnen sie zu senden. Wenn das mehrere Teilnehmer gleichzeitig machen (MA), entsteht eine Konfliktsituation. Im Gegensatz zum Ethernet, beim dem das gleichzeitige Senden zur Zerstörung der beteiligten Telegramme führt, löst das CAN-Protokoll diesen Konflikt zerstörungsfrei auf (CA). Dazu arbeitet der CAN-Bus mit dominanten und rezessiven Buszuständen. Gegebenenfalls überschreibt die dominante, niederohmige „0“ die rezessive, hochohmige „1“. Auf diese Weise entsteht eine Priorisierung der Nachrichten. Die Nachricht mit dem „Identifier“ 0 hat die höchste und die Nachricht dem „Identifier“ 2031 bzw. 536870911 die niedrigste Priorität. Einen Konflikt erkennen die Teilnehmer, indem sie beim Senden am Bus mithören und einen Bitvergleich durchführen (bitorientiertes Protokoll). Bei einer Differenz ziehen die betroffenen Teilnehmer ihre Telegramme sofort zurück. Den nächsten Sendeversuch starten sie, sobald der Bus wieder frei ist. Auf diese Weise setzt sich immer die jeweils höchstpriore Nachricht durch.

[42] CANopen-Safety ist eine standardisierte Lösung von der CiA mit prinzipieller Zustimmung der BIA.

Die nieder- und hochohmigen Buszustände werden über eine „Wired-AND"-Technik realisiert. Aus Gründen der Störsicherheit arbeitet der CAN-Bus jedoch mit einer differentiellen Übertragung, weshalb die Bustreiber zwei gegensinnige „Open-Collector"-Endstufen besitzen. Das Übertragungsmedium ist eine Zweidrahtleitung, welche linienförmig verlegt und an beiden Enden mit ca. 120 Ω abgeschlossen wird[43]. Aus EMV-Gründen besteht der Leitungsabschluss oft aus zwei gleichen Widerständen in Serie, mit einem kapazitiv auf Masse gezogenem Mittelpunkt. Die Baudrate reicht von 10 kBit/s bis 1 MBit/s, wobei die max. Netzausdehnung von 5000 m auf ca. 25 m schrumpft [28]. Aufgrund der Kabeldämpfung, werden für Längen über 1000 m i.A. „Repeater" benötigt. Bei hohen Baudraten jedoch reduzieren sie die Netzausdehnung. Die „Repeater" erhöhen die Signallaufzeit, für die beim CAN-Bus eine obere Schranke von ca. $0{,}25 \cdot T_{Bit}$ existiert. Trotzdem werden manchmal „Repeater" auch bei hohen Baudraten eingesetzt. Und zwar, wenn die max. Teilnehmerzahl von 32 überschritten wird.

Telegramm

Das CAN-Protokoll definiert vier Telegramme: ein Datentelegramm („Data-Frame"), ein Datenanforderungstelegramm („Remote-Frame"), ein Fehlertelegramm („Error-Frame") und ein Überlasttelegramm („Overload-Frame"). Für alle Telegramme gilt die Bitreihefolge „MSB first". Das wichtigste und häufigste Telegramm ist das Datentelegramm. Es wird in der Regel auf Eigeninitiative eines Teilnehmers (z.B. „Interrupt") verschickt und unterstreicht damit den Ereignischarakter von CAN. Seinen Aufbau nach Spezifikation 2.0 A (Standard) zeigt Bild 2.14 – auf Version 2.0 B („Extended") wird nicht eingegangen, der Aufbau ist jedoch ähnlich.

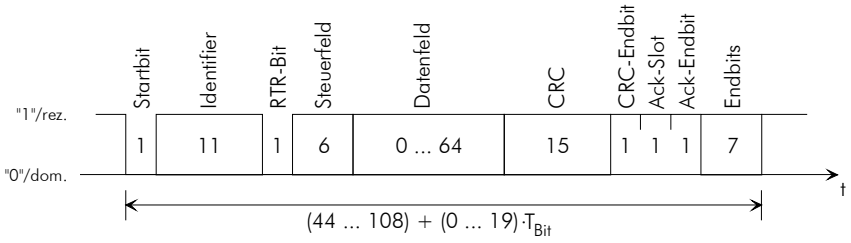


Bild 2.14: Datentelegramm von CAN (2.0 A)

Das Datentelegramm beginnt mit einem dominanten Startbit, gefolgt von einem elf Bit langen „Identifier". Theoretisch existieren $2^{11} = 2048$ „Identifier". Da die Nummern 0x7F0..0x7FF gesperrt sind, verbleiben nur die erwähnten 2032 „Identifier". Als nächstes folgt das „Remote-Transmission-Request-Bit" (RTR-Bit). Es ist beim Datentelegramm stets dominant („0"). Mit diesem Bit endet die Arbitrierungsphase. Das heißt im Konkurrenzfall ist spätestens hier klar, welches Telegramm die höchste Priorität hat[44]. An das RTR-Bit grenzt ein sechs Bit langes Steuerfeld. Die ersten beiden Bits (IDE, r0) werden dominant übertragen. Das IDE-Bit spielt für den gemischten „Standard/Extended"-Betrieb eine Rolle, wohingegen das r0-Bit für spätere Zwecke reserviert ist. In den letzten vier Bits des Steuerfelds wird die Länge des anschließenden Datenfeldes binär codiert. Von den 16 möglichen Nummern sind lediglich die von 0..8 definiert, weil das Datenfeld byteweise belegt wird und max. 8 Bytes umfasst. An das Datenfeld hängt das CAN-Protokoll ein 15 Bit langes CRC-Prüfwort zur Fehlersicherung. Es wird aus allen vorangehenden Telegrammbits (Startbit bis letztes Datenbit) und dem Generatorpolynom aus Formel 2.7 berechnet (siehe S. 46). Die 15 CRC-Bits werden vom Sender mit

[43] Ehemals Daimler-Benz hat eine Sternpunktverdrahtung patentiert [31].

[44] Bei „Extended-CAN" dauert die Arbitrierung etwas länger. Anstelle von 13 Bitzeiten werden dort 33 benötigt [51].

einem rezessiven Bit abgeschlossen. Im Fehlerfall sendet ein oder senden mehrere Empfänger, ab diesem so genannten CRC-Endbit, ein dominantes Fehlertelegramm, wodurch alle betroffenen Teilnehmern davon erfahren.

Im Normalfall verschickt der Sender zwei weitere rezessive Bits, das „Ack-Slot-“ und „Ack-Endbit“. Ersteres dient dem Sender als Empfangsbestätigung. Jeder Teilnehmer, der das Telegramm bis dorthin korrekt empfangen hat, überschreibt das „Ack-Slot-Bit“. Wenn das „Ack-Slot-Bit“ rezessiv bleibt (z.B. in Folge eines Kabelbruchs), ist das ein Fehlersignal für den Sender. Nun besteht jedoch die Möglichkeit, dass auch ein Fehlertelegramm ein dominantes „Ack-Slot-Bit“ erzeugt. Aus diesem Grund existiert das zweite rezessive Bit, das „Ack-Endbit“. Es unterscheidet die beiden Fälle, indem es im Erfolgsfall rezessiv bleibt und bei einem Fehlertelegramm dominant wird. Den Abschluss eines Datentelegramms bilden sieben rezessive „Endbits“. Somit umfasst ein Datentelegramm 44 Bits, wenn das Datenfeld leer ist und 108 Bits, wenn es voll ist.

Bei der gewählten NRZ-Codierung können lange „0“- oder „1“-Folgen entstehen, weshalb der CAN-Bus mit „Bitstuffing“ arbeitet. Der Sender fügt vor jedem sechsten gleichen Bit in Folge ein „Stuffbit“ ein, so dass max. fünf gleiche Bits nacheinander auftreten können. Wie viele „Stuffbits“ pro Telegramm eingefügt werden, hängt vom „Identifier“ und Datenfeld ab. Mehr als 19 Stück können es aber nicht sein. Aus diesem Grund schwankt die Datentelegrammlänge zwischen 44..127 Bits. Jetzt lässt sich auch die Anzahl der „Endbits“ erklären. Sieben Stück hintereinander verletzen das „Bitstuffing“ und sind eindeutig identifizierbar. Allerdings handhaben Empfänger und Sender das ein bisschen unterschiedlich. Auf der Empfängerseite wird eine Nachricht nach dem sechsten rezessiven „Endbit“ als korrekt angesehen. Der Sender hingegen muss bis nach dem Siebten warten. Ein byzantinischer Fehler im sechsten „Endbit“ führt nämlich erst im siebten zu einem Fehlertelegramm.

Das nächste Telegramm, das Datenanforderungstelegramm („Remote-Frame“), ist dem Datentelegramm sehr ähnlich. Ein Datenanforderungstelegramm hat ein rezessives RTR-Bit und enthält keine Daten. Es fordert die Daten in dem Datentelegramm mit dem gleichen „Identifier“ an. Gegenüber dem Datentelegramm ist das Datenanforderungstelegramm, aufgrund des rezessiven RTR-Bits, niedriger priorisiert. Sollten also Anforderung und Antwort zusammentreffen, dann setzt sich die Antwort sinnigerweise durch.

Als Pendant zum Datenanforderungstelegramm existiert das Überlasttelegramm („Overload-Frame“). Es verzögert Daten- und Datenanforderungstelegramme, wodurch sich ein überlasteter Teilnehmer etwas Luft verschaffen kann. Beginnen darf ein Überlasttelegramm nur im ersten Pausenbit des „Interframe-Spaces“. Der Aufbau unterscheidet sich gänzlich von den bisherigen Telegrammen. Ein Überlasttelegramm fängt mit sechs dominanten Bits an und verletzt dadurch die „Stuffbit-Regel“. Auf diese Weise vermeidet man, dass ein verspätetes Überlasttelegramm nicht als Daten- oder Datenanforderungstelegramm missinterpretiert wird. Nach den sechs dominanten Bits sendet der überlastete Teilnehmer acht rezessive Bits. Das erste rezessive Bit wird jedoch überschrieben, da die anderen Teilnehmer als Folge eines Überlasttelegramms auch ein, um ein Bit versetztes, Überlasttelegramm abschicken. Die Gesamtlänge von 14 Bits bleibt jedoch konstant, weshalb man auf dem Bus sieben dominante und sieben rezessive Bits sieht. Maximal sind zwei Überlasttelegramme nacheinander erlaubt.

Schließlich das Fehlertelegramm („Error-Frame“). Es ist vom Aufbau her mit dem Überlasttelegramm gleich. Ein Teilnehmer sendet es eine Bitzeit nach einem erkannten Fehler. Wie erwähnt, verletzen die sechs einläutenden, dominanten Bits die „Stuffbit“-Regel. Daraufhin erkennen alle anderen Teilnehmer den Fehler auch und senden ebenfalls ein Fehlertelegramm. Hierbei kann die dominante Bitfolge bis auf zwölf Stück anwachsen. Aufgrund der erzwungenen Verfälschung, verwerfen alle Teilnehmer das betroffene, fehlerhafte Telegramm. Im Gegensatz zum Überlasttelegramm ändert sich die Anzahl

der rezessiven Bits beim Fehlertelegramm nicht. Deshalb kann ein Fehlertelegramm 14..20 Bitzeiten dauern.

Zeitverhalten

Das Zeitverhalten des CAN-Busses ist sehr komplex. Niederpriore Nachrichten können blockiert werden. Bei stark unterschiedlich langen Telegrammen besteht z.B. die Möglichkeit eines großen Abtast-„Jitters“. Telegrammwiederholungen im Fehlerfall machen eine Übertragungszeitvorhersage nahezu unmöglich. Außerdem hängt das Sende- und Empfangsverhalten eines Teilnehmers von seiner Vorgeschichte ab. Beim CAN-Bus gibt es drei Zustände, die einen Teilnehmer, je nach Fehlerhäufigkeit, Stück für Stück abschalten. „Interframe-Spaces“ dürfen des Weiteren beliebig lang dauern. Und schließlich sind Nachrichten bzw. Telegramm i.A. an äußere Ereignisse („Interrupts“) gebunden. Diese Mechanismen sind durchaus sinnvoll. Sie machen den CAN-Bus flexibel und anwenderfreundlich. Allerdings leidet das Realzeitverhalten darunter. Theoretisch könnte man es für einen Anwendungsfall zwar analysieren, praktisch ist das aber ohne Bedeutung, weil der Aufwand und die Varianz sehr groß sind.

Aus diesem Grund werden Annahmen getroffen, um das Zeitverhalten abschätzen zu können. Für die Datenübertragung kommen nur die „Data-Frames“ in Frage. Da es sich um eine Abschätzung im Sensor/Aktorbereich handeln soll, wird die Anzahl der Datenbytes pro Telegramm mit eins festgelegt. Entsprechend gering fällt die Anzahl der „Stuffbits“ aus. Im Mittel wird ein Telegramm ungefähr zwei „Stuffbit“ aufweisen, woraus sich eine durchschnittliche Telegrammlänge von 54 Bits errechnet. Weiterhin wird ein ereignisgesteuerter Busbetrieb angenommen. Damit bei dieser CAN-typischen Betriebsart wichtige Ereignisse und Daten nicht lange warten müssen, sollen sie in Telegrammen hoher Priorität versendet werden. Mit einer Verzögerung von einem Telegramm muss immer gerechnet werden, weil die Telegramme nicht unterbrechbar sind. Je nach Priorität, kann es aber auch zu längeren Wartezeiten kommen. Bei einer moderaten Buslast, wird sich eine hochpriorisierte Nachricht jedoch nie länger als um zwei oder drei Telegramme verzögern.

Als Baudrate soll 500 kBit/s verwendet werden. Dieser Wert hat sich im Kfz und in der Industrie als noch unproblematisch erwiesen. Im Kfz werden die Teilnehmer meist über einige Meter lange, offene Stichleitungen an den CAN-Bus angeschlossen. Dadurch entstehen Reflexionen, die bei 1 MBit/s vielfach zu hoch sind. Aus diesem Grund arbeiten viele CAN-Busse in Kfzs nicht mit der max. Baudrate. Außerdem ist die Netzausdehnung bei 500 kBit/s ungefähr drei- bis viermal so groß wie bei 1 MBit/s. Dies könnte auch der Grund für die Beschränkung der Baudrate auf 500 kBit/s beim CAN-basierten SafetyBus sein.

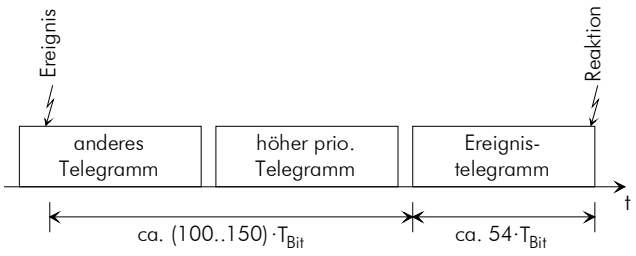


Bild 2.15: Ungefähre Reaktionszeit bei hochpriorisierten CAN-Nachrichten

Mit diesen Zahlenwerten lässt sich die Reaktionszeit bei hoch priorisierten Nachrichten abschätzen. Gemäß Bild 2.15 liegt sie bei ca. 200..300 µs im fehlerfreien Fall. Allerdings kann man die Reaktionszeit nicht garantieren. Dafür gibt es zu viele Wenn und Aber, wobei die erwähnten Telegrammwieder-

holungen das größte Realzeitproblem darstellen. Aus diesem Grund hat man für harte Realzeitsysteme TTCAN entwickelt. In dieser CAN-Variante ist der Sendewiederholmechanismus deaktivierbar. Außerdem hat man einen „Session Layer“ (OSI 5) hinzugefügt, der eine Zeitsteuerung und Synchronisation enthält [29].

In einem TTCAN-Netz gibt es einen Zeit-„Master“, der alle anderen Teilnehmer über ein bestimmtes „Frame“ periodisch synchronisiert. Nach jedem dieser „Frames“ beginnen die Teilnehmer geordnet in „Slots“ zu senden. Für Sensor/Aktoranwendungen kann man die Dauer der „Slots“ näherungsweise berechnen. Ausgangspunkt ist wieder ein Datentelegramm mit einem Datenbyte. Unter Berücksichtigung von durchschnittlich zwei „Stuffbits“ pro Telegramm und mindestens drei Bitzeiten Pause nach einem Telegramm, beläuft sich die „Slot“-Dauer auf etwa 57 Bitzeiten (siehe Bild 2.16). Das ergibt bei einer Baudrate von 1 MBit/s die sehr kurze Telegrammperiode von 57 µs. Wenn man des Weiteren ca. 20 gleich abgetastete Sensoren und Aktoren annimmt, so gelangt man zu einer Abtastzeiten von ca. 1,1 ms. Abschließend kann man also festhalten, dass mit dem „normalen“ CAN-Bus „weiche“ Realzeitsysteme im oberen µs-Bereich realisierbar sind und mit dem TTCAN-Bus „harte“ Realzeitsysteme im unteren ms-Bereich.

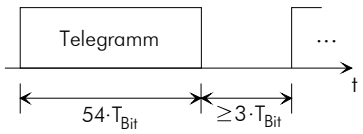


Bild 2.16: Geschätzte Telegrammperiode für Sensoren und Aktoren mit TTCAN

Effizienz

Mit den aufgezählten Einschränkungen ist es zudem möglich, die Übertragungseffizienz für Sensor/Aktoranwendungen abzuschätzen. Die Telegrammeffizienz beträgt, 8 Bit/54 Bit ≈ 15 %. Dieser Wert bildet eine Supremum für die Gesamteffizienz. Aufgrund der möglichen, langen Telegrammpausen kann die Gesamteffizienz deutlich darunter liegen. Für eine relevante Abschätzung darf man nicht mit der minimalen Telegrammpause von $3 \cdot T_{Bit}$ rechnen. Denn der CAN-Bus soll schnell reagieren können, weshalb die Buslast weit unter 100% liegen muss[45]. Aus diesem Grund wird eine 60 %-ige Auslastung angesetzt, bei der im Mittel nach jedem zweiten Telegramm Platz für ein wartendes Telegramm ist. Daraus resultiert für den CAN-Bus bei Sensor/Aktordaten eine Gesamteffizienz von ca. 15 %·0,6 = 9 %. Der TTCAN-Bus schneidet hier besser ab. Dank seiner Zeitrestriktionen darf er voll ausgelastet werden (minimale Pausen), weshalb seine Gesamteffizienz ca. 8 Bit/57 Bit ≅ 14 % beträgt.

Verlässlichkeit

Der CAN-Bus enthält fünf Mechanismen zur Erkennung von Telegrammfehlern: „Bit-Monitoring“, „Frame-Check“, CRC-Codierung, „Acknowledgement-Check“ und „Bit-Stuff-Check“. Bis auf die CRC-Sicherung, wurden bereits alle beschrieben. Das CRC-Prüfwort wird dem Generatorpolynom in Formel 2.7 berechnet[46].

Theoretisch stellt sich eine Hamming-Distanz $d_{min} = 6$ ein. Praktisch jedoch beträgt $d_{min} = 2$ [48]. Die Ursachen hierfür sind die „Stuff-“ und „End-of-Frame-Bits“. Das „Bitstuffing“ geschieht nach der CRC-Codierung, wodurch sich die Code-Eigenschaften verändern. So kann es durch die Verfälschung

[45] Bei 100% Buslast könnten z.B. fehlerpassive Teilnehmer nicht mehr in den Normalzustand (fehleraktiv) zurückkehren.

[46] Es ist für „Frames“ mit weniger als 127 Bits gut geeignet (BCH-Code) [27].

eines „Stuffbits“ und eines „normalen“ Bits passieren, dass Letzteres beim „Destuffing“ herausgenommen wird und ein verfälschtes, aber gültiges Telegramm übrig bleibt. Außerdem können zwei Bitfehler bei bestimmten Telegrammen, an den richtigen Stellen, eine Verkürzungen vorspielen. Dadurch entstehen weitere falsche, nicht erkennbare Telegramme[47]. Ob die „Burst“-Fehlererkennung des CRC-Codes auch darunter leidet, ist ungewiss, aber wahrscheinlich.

$$g(x) = x^{15} + x^{14} + x^{10} + x^8 + x^7 + x^4 + x^3 + 1$$

Formel 2.7: CRC-Generatorpolynom für den CAN-Bus [27]

Glücklicherweise gibt es nur wenige Codewörter mit einem Abstand $d = 2$, so dass die Restfehlerwahrscheinlichkeit nicht allzu sehr ansteigt. Unter Beachtung aller Fehlererkennungsmechanismen, wurden in [48] Berechnungen für ein paar Beispiele angestellt. Die wichtigsten Ergebnisse sind in Bild 2.17 dargestellt. Aus den Graphen geht eine Restfehlerwahrscheinlichkeit kleiner 10^{-12} hervor. Sie gilt jedoch nur für die angegebenen Randbedingungen. Allgemein muss man mit der Restfehlerwahrscheinlichkeit von $4{,}7 \cdot 10^{-11}$,aus der Spezifikation, rechnen [27]. Damit erreicht der CAN-Bus leider nur die Datenintegritätsklasse I2 nach DIN 19244 (siehe Abschnitt 8.7).

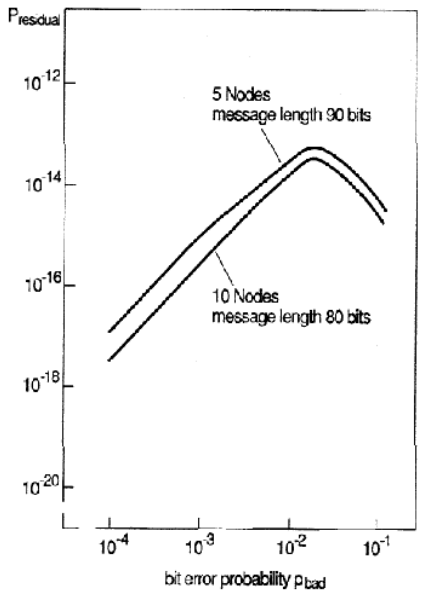


Bild 2.17: Beispiele der Restfehlerwahrscheinlichkeit beim CAN-Bus [48]

Im Vergleich zu vielen anderen Bussystemen besitzt der CAN-Bus eine geringe Restfehlerwahrscheinlichkeit, zumal er auch viele byzantinische Fehler erkennt. Diese lokalen, schwer handhabbaren Fehler werden durch die Fehlertelegramme global gemacht, wodurch sie besser kontrollierbar sind. Außerdem verfügt der CAN-Bus über ein Fehlereingrenzungsverfahren mit drei Zuständen. Im Normalfall befindet sich ein Teilnehmer im Zustand „Error-Aktiv“. Dort kann er, wie beschrieben, am Busverkehr teilnehmen. Registriert oder verursacht ein Teilnehmer mehr als 127 Fehler in einem gewissen Zeitraum, so wechselt er in den Zustand „Error-Passiv“. Es besteht die Möglichkeit, dass der

[47] Laut [48] führen diese Fehler beim amerikanischen J1850-Bus zu einer Hamming-Distanz von eins.

Teilnehmer sporadische Defekte aufweist. Aus diesem Grund werden seine Busaktivitäten eingeschränkt. Vor einer Arbitrierung muss er länger warten als intakte Teilnehmer. Zudem kann er nur modifizierte Fehlertelegramme mit rezessiven Pegeln senden („Passive-Error-Frame“). Mit diesem passiven Verhalten wird einem Teilnehmer die Möglichkeit entzogen den CAN-Bus häufig zu stören.

Sollte sich ein Teilnehmer im Zustand „Error-Passiv“ wieder bewähren, so kann er in den Normalzustand („Error-Acitv“) zurückkehren. Die Entscheidung trifft ein „Transmit-Error-Counter“ (TEC) und ein „Receive-Error-Counter“ (REC). Wenn beide einen Wert unterhalb von 127 anzeigen, geschieht dieser Zustandswechsel; den Wechsel nach „Error-Passiv“ leitet bereits ein einziger Zählerstand > 127 ein. Damit dieser Mechanismus funktioniert, müssen Fehler und Erfolge gegeneinander aufgewogen werden. Prinzipiell führt ein Fehler zur Erhöhung und ein Erfolg zur Erniedrigung des jeweiligen Zählers. Die In- und Dekremente sind jedoch unterschiedlich, weil die Fehler über acht Regeln gewichtet werden.

Der dritte und letzte Zustand ist „Bus-Off“. Er wird eingenommen, falls der TEC eines Teilnehmers den Wert 255 übersteigt. In diesem Zustand schaltet sich ein Teilnehmer komplett vom Bus weg. Denn die Wahrscheinlichkeit für einen dauerhaften Defekt ist hier groß. Für den Übergang in den Normalzustand wird ein „Reset“ benötigt.

Trotz aller Schutzmechanismen bildet die min. Codedistanz $d_{min} = 2$ eine Schwachstelle im CAN-Bus. Sie ist zwar klein, aber vorhanden und macht den CAN-Bus in bestimmten Situationen fehleranfällig. In [34] wird der Vorschlag gemacht, die „Stuffbits“-produzierenden „Identifier“ wegzulassen. Das Problem wird dadurch nicht beseitigt, weil „Stuffbits“ auch durch Daten entstehen können. Selbst wenn man keine Daten überträgt, ist das Problem nicht gelöst – sofern das überhaupt ein Lösungsvorschlag ist. Denn es existieren noch die unerkannten, verkürzten Telegramme. Ohne Protokolleingriffe kann man an der minimalen Codedistanz $d_{min} = 2$ nichts ändern. Ein Eingriff in das Protokoll ist aufgrund der Etabliertheit des CAN-Busses nicht möglich.

2.4 Zusammenfassung

Sensor/Aktorbusse sind Bussysteme für die Vernetzung von Schaltern, Lampen, Temperaturfühlern, Stellmotoren, Relais, Feuchtigkeitsfühlern, Heizungen, „Airbags“ usw. Sie müssen kostengünstig sein, damit sich ihr Einsatz auch aus wirtschaftlichen Gesichtspunkten lohnt. Die Datenpakete von Sensor/Aktorbussen sind klein, typischerweise 1..16 Bits. Dafür sind die zeitlichen Anforderungen relativ hoch. Gewöhnlich werden Senor/Aktordaten alle 1..10 ms aktualisiert. Weitere Charakteristika von Sensor/Aktorbussen sind die Lokalität und die geringe Teilnehmerzahl. Die örtliche Ausdehnung liegt etwa bei 10..100 m und die maximale Teilnehmerzahl beträgt in der Praxis zwischen 30 und 40.

Aufgrund der gewachsenen Systemkomplexität, neuen Anwendungsgebieten, dem Wunsch nach intelligenten Systemen und dem Wunsch nach durchgängiger Vernetzung, haben Sensor/Aktorbusse an Bedeutung gewonnen. Gleichzeitig sind die Anforderungen an diese Bussysteme gestiegen. Sensor/Aktorbusse müssen heute äußerst kostengünstig sein, zunehmend „harte“ Realzeitrestriktionen erfüllen können, den gestiegenen Sicherheitsansprüchen gerecht werden und teilweise über höhere Bandbreiten verfügen. Dass die existierenden Sensor/Aktorbusse oder vergleichbare Busse dafür immer weniger geeignet sind, zeigen die aktuellen, großen Anstrengungen in der Forschung und Entwicklung. Es werden etablierte Bussysteme ergänzt oder modifiziert und neue Bussysteme geschaffen. Allerdings mit mehr oder minder Erfolg, vor allem bei den Eigenschaften Realzeitfähigkeit, Effizienz und Verlässlichkeit. Das zeigt die Analyse und Bewertung insbesondere mit TTP/A vergleichbaren Bussen, die in dieser Arbeit entstand. Momentan ist TTP/A der einzige Sensor/Aktorbus, der die Anforderungen vieler und unterschiedlicher Anwendungen zumindest konzeptionell erfüllt.

3 Sensor/Aktorbus TTP/A

3.1 TTP/A-Architektur

Bei der Konzipierung von TTP/A wurde ein für Bussysteme eher seltener Weg eingeschlagen. Wie Bild 3.1 zeigt, besteht das TTP/A-Protokoll aus drei Bausteinen: „Basic-TTP/A“, „Extended-Reliability“ und „Interface-File-System“.

„Basic-TTP/A“ zeichnet für die grundlegenden Eigenschaften verantwortlich. Im Wesentlichen ist das die realzeitfähige, effiziente, serielle Kommunikation zwischen den Knoten. Außerdem sorgt „Basic-TTP/A“ für die Entkopplung der anderen TTP/A-Bausteine und der Applikation von der Hardware. Das Einsatzgebiet sind einfache, kostengünstige Anwendungen mit und ohne Realzeitanforderungen. Für sicherheitskritische Systeme ist „Basic-TTP/A“ nicht geeignet. Allerdings erfüllt es einige Voraussetzung dafür, so dass dieses Gebiet, durch Hinzufügen von Sicherheits- bzw. Verlässlichkeitsmerkmalen, auch erschließbar ist. Genau das ist die Aufgabe des TTP/A-Bausteins „Extended-Reliability“. Eine Erweiterung von „Basic-TTP/A“ hinsichtlich Flexibilität, Komfort und einheitlicher Informationsdarstellung bietet der Letzte von den drei TTP/A-Bausteinen. Das so genannte „Interface-File-System“ (IFS) macht den TTP/A-Bus „Plug&Play“-fähig und verleiht ihm die Möglichkeit zur „Online“-Konfiguration bzw. -Wartung. Außerdem stellt es dem Anwender sämtliche Informationen in standardisierter Form dar [17]. Die Analogie der TTP/A-Bausteine zum OSI-Referenzmodell (Open Systems Interconnection) findet man in den Klammern in Bild 3.1.

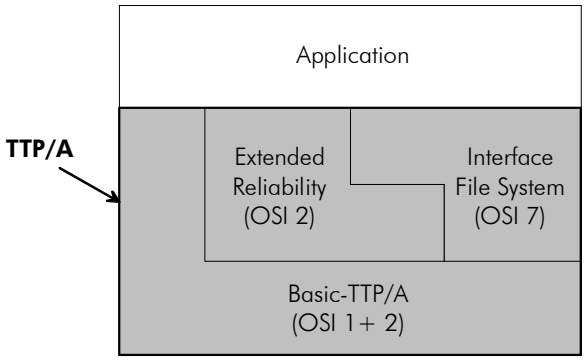


Bild 3.1: TTP/A-Architektur

Aufgrund der Gliederung ist TTP/A skalierbar, wodurch es „Overhead“-arm auf verschiedene Anwendungen angepasst werden kann. Für µCs, DSPs oder „Embedded-Systems“ ist das wichtig. Ihre Ressourcen fallen sehr unterschiedlich aus. Das Spektrum reicht von leistungsstarken 32 Bit-Maschinen mit viel Peripherie und Speicher im MB-Bereich, bis hin zu 4 Bit-µCs mit mäßiger Rechenleistung,

wenig Peripherie und einigen KB Speicherplatz. Ohne Skalierbarkeit könnte man nur den oberen Teil bedienen. Attraktiver sind jedoch die günstigen, kleinen µCs aus der PIC- oder AVR-Familie, um zwei Beispiele zu nennen, mit einem Preis bei hohen Stückzahlen in der Größenordnung von 0,5..1 €. Ihre begrenzten Ressourcen schränken die Funktionalität von TTP/A ein. Deshalb ist „Basic-TTP/A“ der einzig obligatorische TTP/A-Baustein. Zusammen mit einer Applikation, reicht das für viele einfache, kostensensitive Anwendungen vollends aus. Optional sind hingegen die beide anderen Bausteine „Extended-Reliability“ und „Interface-File-System“. Dadurch entstehen $2^2 = 4$ Kombinationen, wie Tabelle 3.1 zeigt:

TTP/A-Bausteine	**Einsatzgebiet**
„Basic-TTP/A“	einfache, günstige Anwendungen mit und ohne Realzeitanforderungen (Drehzahlregelung, Blinker, Fensterheber etc.)
„Basic-TTP/A“ + “Extended-Reliability”	sicherheitskritische, kostensensitive Anwendungen (Notaus-Steuerung, „Airbag“-Zündung etc.)
„Basic-TTP/A“ + “Interface-File-System”	flexible, günstige Anwendungen mit und ohne Realzeitanforderungen (Internet-fähige Steuerkonsole)
„Basic-TTP/A“ + “Extended-Reliability” + “Interface-File-System”	sicherheitskritische, flexible Anwendungen mit Augenmerk auf die Kosten (konfigurierbare, medizinische Prothesen)

Tabelle 3.1: Skalierbarkeit von TTP/A

Ein weiterer Vorteil der TTP/A-Architektur ist die Offenheit. Mit den vier Kombinationen kann man viele Einsatzfälle abdecken. Für Spezialfälle sieht die TTP/A-Architektur anwenderspezifische Erweiterungen oberhalb von „Basic-TTP/A“ vor. So kann man z.B. anstelle des „Interface-File-Systems“ ein eigenes „Gateway“ verwenden oder den Baustein „Extended-Reliability“ durch eine „Forward-Error-Correction“ ersetzen. Der Wunsch nach dieser Möglichkeit ist vorhanden. Man erkennt das am Trend hin zu den Zeitsteuerungen für realzeitfähige Bussysteme. Viele Busse besitzen diese Eigenschaft nicht von Haus aus. Wegen des üblichen, monolithischen Aufbaus, lassen sie keine Protokollanpassungen zu. Deshalb versuchen viele Anwender Zeitsteuerungen in den Applikationen zu realisieren. In der Regel funktionieren diese zufriedenstellend, solange keine Übertragungsfehler auftreten. Dann jedoch geht die Realzeitfähigkeit meistens verloren, beispielsweise durch die geschilderten automatischen Telegrammwiederholungen.

TTP/A ist aufgrund seiner Architektur für Neuerungen und Innovationen offen. Begünstigt wird dieser Ansatz durch die Tatsache, dass das Protokoll ausschließlich in Software realisiert ist. Außerdem teilt sich TTP/A mit der Applikation einen µC, DSPs o.Ä. Deshalb kann ein Anwender mit den gleichen Werkzeugen, mit denen er die Applikation erstellt, auch TTP/A skalieren und/oder erweitern. Er muss dazu nur die gewünschten, separat übersetzten Software-Teile („Object-Files“) im Bindevorgang zusammenfügen. Dieser Ansatz ist nicht neu. Er wird bei vielen Betriebssystemen, insbesondere bei den Realzeitvarianten, seit langem erfolgreich angewandt (VxWorks, RTEMS, µCOS, Linux[48]...). Man kann TTP/A teilweise mit einem kleinen, skalierbaren Realzeitbetriebssystem vergleichen – das Basisprotokoll „Basic-TTP/A“ steht dem Betriebssystemkern („Micro-Kernel“) gegenüber, der Baustein „Extended-Reliability“ einem „Error-Manager“ und das „Interface-File-System“ einem „I/O-Manager“.

[48] Linux belegt, dass die Ängste bezüglich eines Wildwuchses unbegründet sind. Die Offenheit hat sehr zur Weiterentwicklung und Verbreitung von Linux beigetragen.

3.2 Basisprotokoll „Basic-TTP/A“

3.2.1 Topologie

„Basic-TTP/A“ ist für den Gebrauch von preiswerten Standardbauteilen konzipiert. Entsprechend einfach fällt die Topologie aus. Bild 3.2 zeigt sie anhand eines Beispielnetzes. Es handelt sich um die gängige Linienstruktur, da es hierfür jede Menge bewährte und günstige „Transceiver“ (ISO 9141, RS485, CAN...) gibt. An diesen linienförmigen Bus werden ein „Master“ und n „Slaves“ angeschlossen. Der „Master“ verfügt über den stärksten Rechner und die meisten Ressourcen. Bei den „Slaves“ hingegen beschränkt man sich auf das Nötigste, weil sie mit dem Faktor n in die Gesamtkosten eingehen. Die Obergrenze n_{max} für die Anzahl der „Slaves“ hängt vom verwendeten „Physical-Layer“ ab. „Basic-TTP/A“ setzt hier keine Schranken, da es weder mit Telegrammnummer noch mit Knotenadressen arbeitet. Mit z.B. RS485-„Transceivern“ ist $n_{max} = 32$. Abzüglich des „Masters“ verbleiben damit max. 31 „Slaves“. Durch den Einsatz von „Repeatern“ lässt sich die Anzahl vergrößern. Mehr Details zum „Physical-Layer“ enthält Abschnitt 3.3.

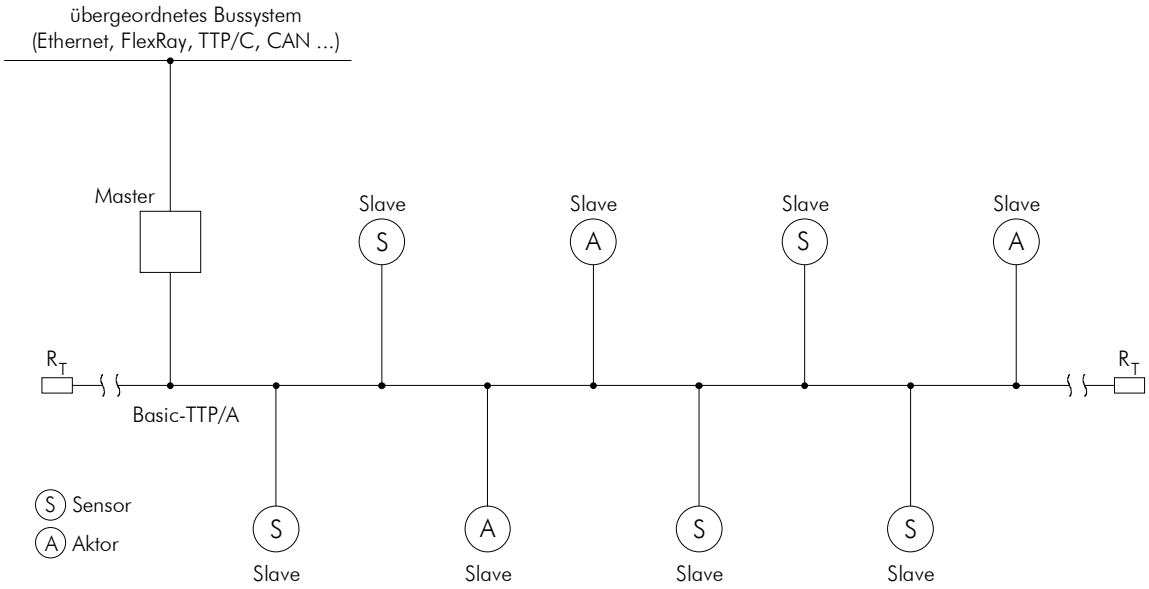


Bild 3.2: Topologie eines „Basic-TTP/A“-Netzes

Außerdem kann man „Basic-TTP/A“ als „Stand Alone“- oder „Sub“-Bus betreiben. Im letzteren Fall übernimmt der „Master“ zusätzlich die Funktion eines „Gateways“. Topologisch gesehen ist ein „Basic-TTP/A-Sub“-Bus weiterhin eine Linie. Die Gesamtstruktur ist dann jedoch ein Baum mit „Basic-TTP/A“-Ästen. Eine zentrale Rolle spielt in beiden Betriebsarten der „Master“. Er ist eine so genannte „Golden-Unit“. Wenn der „Master“ ausfällt, steht das ganze („Sub“-)System. Aus diesem Grund kann man ohne zustätzliche Maßnahmen mit „Basic-TTP/A“ keine verlässlichen Systeme aufbauen. Hierfür gibt es den TTP/A-Baustein „Extended-Reliability“. Er erhöht die Übertragungssicherheit und sieht redundante Strukturen, wie z.B. einen Schatten-„Master“ oder ein Parallelnetz, vor (siehe Kapitel 6). Dadurch wird die Topologie erweitert, aber prinzipiell nicht geändert.

3.2.2 Kommunikationsprinzip

Das „Basic-TTP/A“-Protokoll ist eine Kombination aus dem TDMA- und „Master/Slave“-Prinzip. Beide Arbitrierungsarten sind realzeitfähig und ressourcenarm umsetzbar. Unterschiede bestehen bei der Flexibilität und Effizienz. Klassische „Master/Slave“-Busse sind flexibel, aber ineffizient. Dagegen sind reine TDMA-Busse effizient, aber starr. Durch die Vermischung der beiden Methoden kann man die Vorteile beider Verfahren nutzen, wodurch eine flexible, effiziente, realzeitfähige und ressourcenarme Datenübertragung möglich ist. Dazu werden die Daten blockweise transportiert. Innerhalb der Blöcke herrscht das TDMA- und außerhalb das „Master/Slave“-Prinzip. Auf diese Weise antwortet auf einen „Master“-Aufruf nicht nur ein „Slave“, sondern es findet eine autarke, zeitlich begrenzte Kommunikation zwischen den Knoten statt[49]. Wegen der TDMA-Terminologie, heißen die Datenblöcke hier Runden.

„Master/Slave“-TDMA-Arbitrierung

Bei „Basic-TTP/A“ werden, gemäß Bild 3.3 oben, zwei Runden durch eine Pause, die so genannte „Interround-Gap“ (kurz IRG), getrennt. Sie wird für die Synchronisation „bootender Slaves“ benötigt. Die Dauer der IRG könnte etwas kürzer ausfallen als gewählt. Zu Gunsten einfacher, zeitlicher Berechnungen ist sie jedoch genau so lang wie ein „Slot“ innerhalb einer Runde. Die Runden müssen nicht gleich ablaufen. Es gibt 16 verschiedene Runden über deren Reihenfolge die „Master“-Applikation zur Laufzeit entscheidet. Zur Differenzierung besitzen die Runden eine Nummer zwischen 0..15. Typisch für „Basic-TTP/A“ sind periodische Rundenreihenfolgen. Zum einen machen sie die Daten idempotent und zum anderen müssen viele technische Prozesse so bedient werden (z.B. Abtastregelung). Trotzdem sind auch aperiodische Folgen oder sporadische Runden möglich. Je nachdem, wie die „Master“-Applikation die Runden initiiert. Beispiele sind Ereignisreaktionen (z.B. Notaus) oder Konfigurations- und Diagnosezyklen durch ein übergeordnetes Bussystem.

Diese Flexibilität gibt es in den Runden nicht, weil hier das TDMA-Prinzip herrscht. In Bild 3.3 Mitte sieht man, dass die Runden in äquidistante Zeitschlitze (kurz „Slots“) unterteilt sind. In den 13 Bitzeiten dauernden „Slots“ befindet sich je ein elf Bit langer UART-„Frame“. Die nominell zwei Bitzeiten lange Pause, mit dem Namen „Interframe-Gap“ (kurz IFG), wird für die Kompensation von unvermeidlichen Zeitfehlern benötigt. Diese entstehen aufgrund der verteilten Uhren. Der Ablauf in den Runden wird mit vorab erstellten „Schedules“ festgelegt. Hierdurch weiß jeder Knoten, in welchen Runden er wann senden, zuhören oder nichts tun muss. Aus diesem Grund können die Knoten eine Runde selbstständig zu Ende führen, wenn sie der „Master“ einmal angestoßen hat. Die Effizienz bei dieser Art der Kommunikation ist hoch. Allerdings auf Kosten der Flexibilität in den Runden. Nur mit „Basic-TTP/A“ können die „Schedules“ nicht geändert werden. „Basic-TTP/A“ erlaubt aber das Anpassen der „Schedules“ im Betrieb. Deshalb kann man entweder mit dem TTP/A-Baustein IFS oder entsprechenden Applikationen flexibel agieren. In Summe gesehen ist das „Basic-TTP/A“-Protokoll nicht ganz so flexibel wie ein „Master/Slave“- und nicht ganz so effizient wie ein TDMA-Bus.

Aufgrund der 8 Bit-Architektur vieler „low-cost“ µCs und der Übertragung von Daten zu je 8 Bit mit einem UART, wurde festgelegt, dass „Basic-TTP/A“ byteorientiert arbeiten soll. Eine Folge daraus ist die Unterscheidbarkeit von maximal 256 „Slots“ – theoretisch sind natürlich mehr möglich. Da ein „Slot“ für die IRG benötigt wird, verbleiben maximal 255 „Slots“ pro Runde. Die Nummerierung der „Slots“ in einer Runde beginnt bei 0 und endet bei 254. In der Praxis wird man die volle Rundelänge selten ausschöpfen. Trotzdem ist es wichtig die maximale „Slot“-Anzahl im Konzept zu berücksichti-

[49] In Stücken ähnelt das „Basic-TTP/A“-Prinzip dem „Producer-Consumer“-Verfahren vom WorldFip-Bus (Fa. Telemecanique) und dem Summenrahmen-Verfahren des Interbus-S (Fa. Phoenix).

gen, weil eine Reduzierung wesentlich leichter durchführbar ist als eine Erhöhung. Dem ersten „Slot“ einer Runde („Slot“ 0) kommt eine besondere Bedeutung zu. Er enthält den so genannten „Fireworks-Frame“, in dem die Nummer der aktuellen Runde codiert ist. Der „Fireworks-Frame“ wird stets vom „Master“ gesendet und von allen „Slaves“ ausgewertet. Alle anderen „Slots“ enthalten UART-„Frames“ mit Daten, über deren Quelle und Senke(n) der Anwender mit den „Schedules“ frei verfügen kann. Nach dem Empfang eines „Fireworks-Frames“ kennen die „Slaves“ die eingeläutete Runde, laden den entsprechenden „Schedule“ und arbeiten ihn selbstständig ab. Mit dem „Fireworks-Frame“ wird also eine kurzzeitige „Kettenreaktion“ bzw. ein kleines „Datenfeuerwerk“ ausgelöst. Weiterhin dienen die „Fireworks-Frames“ als implizite Synchronisationsmarken für die „Slaves“, wodurch keine Effizienzeinbuße durch die notwendige Synchronisation entsteht; Näheres dazu enthalten Abschnitt 3.2.5 und vor allem Kapitel 4.

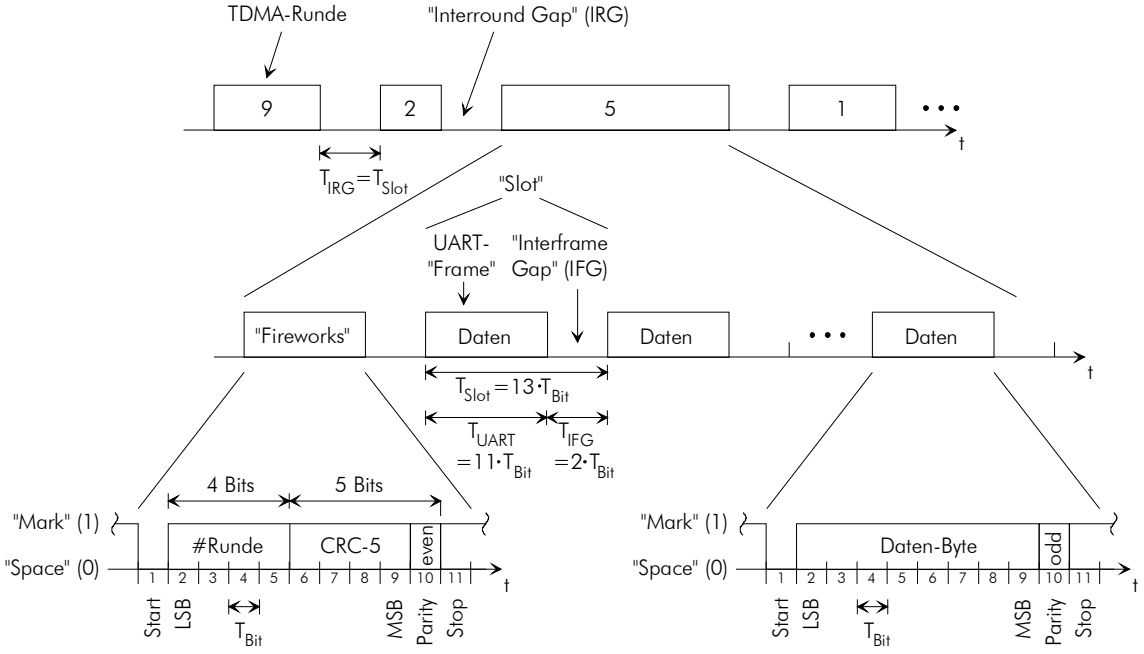


Bild 3.3: Kommunikationsprinzip von „Basic-TTP/A“

Fehlersicherung

Der untere Teil von Bild 3.3 enthält den Aufbau der UART-„Frames“ für die Daten und Rundennummern. Es sind Standard-UART-„Frames“ mit einem Startbit, acht Informationsbits, einem Paritybit und einem Stoppbit. Somit stellt das Byte die atomare Dateneinheit in „Basic-TTP/A“ dar. Um die „Fireworks-Frames“ redundant von den Daten-„Frames“ abzuheben, haben sie eine andere Parität als die Daten-„Frames“. Redundant deshalb, weil „Fireworks-Frames“ durch ihre Position bereits eindeutig bestimmbar sind. Daten-„Frames“ besitzen ein ungerades („odd“) und „Fireworks-Frame“ ein gerades („even“) Paritybit. Bei den Daten genügt diese Fehlersicherung für einfache Anwendungen. Zusammen mit der „Basic-TTP/A“-typischen, periodischen Datenaktualisierung, erfüllen sie die Datenintegritätsklasse I1 nach DIN 19244 (siehe Abschnitt 8.7).

Die Rundennummern müssen hingegen besser abgesichert werden. Sie können auch aperiodisch auftreten, ohne die Idempotenz von wichtigen Daten zu stören. Sporadische Runden, wie im Abschnitt 5.4 an Hand eines Demonetzes realisiert, sind ein Beispiel dafür. Außerdem ist das Gefahrenpotenzial bei einer verfälschten, unerkannten Rundennummer größer als bei einem Datum. Wird ein Datum

verfälscht, so ist die Kommunikation in der Regel zwischen wenigen Knoten kurzzeitig gestört. Eine falsche Rundennummer dagegen, insbesondere bei einem byzantinischen Fehler, kann die Kommunikation zwischen vielen Knoten über einen längeren Zeitraum stören. Deshalb ist die Rundennummer mit einer fünf Bit CRC-Prüfsumme gesichert. Auf den ersten Blick scheint das nicht möglich zu sein, weil die Rundennummer vom UART-Byte bereits vier Bits belegt. Bei geschickter Wahl des Generatorpolynoms entstehen allerdings Codewörter mit gerader Parität, so dass sich das Paritybit des UART-„Frames“ in die Fehlersicherung integrieren lässt. Auf diese Weise konnte die Hamming-Distanz von drei auf vier erhöht und die bessere Datenintegritätsklasse I2 nach DIN 19244 erreicht werden. Bezüglich der Fehlersicherung hinterlässt „Basic-TTP/A“ somit ein harmonisches Bild. Weitere Details befinden sich im Abschnitt 3.2.8 und Restfehlerwahrscheinlichkeitsberechnungen in Kapitel 4. Ein Konzept für eine erweiterte Fehlersicherung enthält Kapitel 6.

Implementierbarkeit und Busphysik

Im Vorgriff sei erwähnt, dass sich dieses Kommunikationsprinzip ressourcenarm und kostengünstig umsetzen lässt. Es konnte gezeigt werden, dass einfache „Basic-TTP/A-Slaves“ mit weniger als 2 KB Speicher auskommen (siehe Kapitel 5). Dadurch eignen sich auch „low-cost“ µCs als „Basic-TTP/A“-Plattform. Der Kostenaspekt schlägt sich aber auch anderweitig nieder. Für die Vernetzung benötigt man lediglich gewöhnliche Kupferdrähte, auf denen die UART-„Frames“ im Basisband, NRZ-codiert übertragen werden. Letzteres bewirkt, dass in „Basic-TTP/A“ Bit- und Baudrate identisch sind[50]. Aus diesem Grund werden diese Begriffe hier gemischt verwendet.

3.2.3 „Round Description List“

Die „Round Descrition List“ (RODL) ist ein abstraktes Gebilde. Formal gesehen enthält sie in irgendeiner Form alle „Scheduling“-Informationen für den korrekten Betrieb eines Knotens. Jeder Knoten hat seine eigene RODL, weshalb in einem „Basic-TTP/A“-Netz genau so viele RODLs existieren wie Knoten. Die „Schedules“ für die Knoten sind ja nicht identisch, auch wenn sie die gleiche Runde betreffen. Wenn z.B. ein Knoten in einem „Slot“ schreibt und ein anderer das lesen möchte, so unterscheiden sich zumindest die „Slot“-Attribute. Außerdem können die „Schedules“ verschieden lang sein, je nachdem wie oft die Knoten an einer Runde partizipieren. Und schließlich gibt es kein festgelegtes RODL-Format. Eine RODL darf sich z.B. in einer verketteten Liste, Tabelle oder im Programmcode befinden. Das hängt von der Implementierung und Optimierung ab.

Vollständige RODL

Zu den wichtigsten RODL-Daten zählen die Arbitrierungsdaten. Im Speziellen sind das die Zeitpunkte und die Art der Buszugriffe. Für die Zeitpunkte sind z.B. „Slot“-Nummern oder „Timer“-Werte denkbar. Unabhängig davon garantieren diese Informationen jedoch keinen kollisionsfreien Busverkehr, da es die in Kapitel 4 erwähnten Uhrenfehler in den „Slaves“ gibt. Aus diesem Grund findet eine Uhrensynchronisation statt, die auf dem Vermessen der „Fireworks“-Abstände fußt. Davon sind insbesondere „Slaves“ mit ungenauen Oszillatoren (RC und „On-Chip“) betroffen. Für eine Korrektur müssen sie die Sollabstände kennen. Deshalb gehören diese oder adäquate Informationen zu den RODL-Daten. Bei „Slaves“ mit Quarzoszillatoren ist die Synchronisation unproblematischer. Trotzdem sollten auch sie den Sollabstand zwischen den „Fireworks-Frames“ in irgendeiner Form speichern. Mit diesen Informationen kann ein „Slave“ mehr Fehler erkennen, was vor allem für eine automatische Baudratenerkennung wichtig ist. Aus dem gleichen Grund müssen auch nicht gebrauchte Runden gekennzeichnet werden (z.B. mit der Länge Null).

[50] $f_{Bit} = f_{Baud} \cdot ld(n)$ mit n: Anzahl der Symbole (hier n = 2).

Für eine vollständig RODL fehlen noch Angaben über die Quellen und Senken der Daten. Zwischen einer Applikation und „Basic-TTP/A“ findet ein Datenaustausch statt. Deshalb müssen beide Teile wissen, wo sie die entsprechenden Daten ablegen und abholen sollen. Es gibt viele Möglichkeiten für die Realisierung dieser Schnittstelle. Einen hohen Komfort bieten z.B. FIFOs, „Message-Boxes“ oder „Queues“. Diese gängigen Mechanismen für die Interprozesskommunikation wirken entkoppelnd und teilweise synchronisierend. Neben dem eigentlichen Speicherplatz, benötigen sie jedoch zusätzlich Speicher und Rechenzeit für Verwaltungsaufgaben. Bei größeren µCs bzw. „Embedded-Systems“ spielt das keine Rolle, anders bei den ressourcenarmen „low-cost“ µCs. Effizienter ist hier „Shared-Memory“. Und zwar ohne Pufferfunktion, damit man keinen Speicher vergeudet. Der gemeinsame Speicher ist Teil der RODL, weshalb er zu „Basic-TTP/A“ gehört. Umgekehrt macht das weniger Sinn, denn es gibt viele Applikationen, die ein beliebiges Datum mit „Basic-TTP/A“ transportieren (z.B. Temperaturwert, Druckwert, Schalterstellung...). Allerdings gibt es nur wenige Applikationen, die exakte das Gleiche machen. Eine Einschränkung entsteht dadurch nicht. Man verwendet anstelle einer Applikationsvariablen lediglich eine „Basic-TTP/A“-Variable. Für Anwendungsfälle mit Synchronisationsbedarf oder speziellen Fehlerstrategien – das gehört zu einer vernünftigen Datenschnittstelle – hält das „Basic-TTP/A“-Konzept einfache, aber wirkungsvolle Mechanismen bereit (siehe Kapitel 5).

Speicherbedarf

Nachdem nun klar ist, welche Informationen eine RODL enthält, wird an ein paar Beispielen gezeigt, warum es kein festes RODL-Format gibt: Ein einfacher Sensor-„Slave“ soll seinen Ein-Byte-Sensorwert in einem „Slot“ verschicken. Für den Sensorwert benötigt er ein „Shared“-Byte im RAM[51]. Außerdem belegen die Informationen über die Rundenlängen und die „Fireworks“-Codes je 16 Bytes im FLASH[52]. Weiterhin soll der Sensor-„Slave“ hoch optimiert sein. Deshalb wird seine Runde und sein „Slot“ im Programm codiert. Gegenüber dem Speicherbedarf für den Programmcode und die sonstigen Variablen, fallen die 33 Bytes für die RODL hier kaum ins Gewicht. Erfahrungsgemäß reichen für solche „Slaves“ gut 1,5 KB FLASH und ca. 10..20 Bytes RAM aus. Als Plattform eignen sich die kleinsten „low-cost“ µCs (≈ ..0,5 €).

Bei einem komfortablen Knoten nimmt die RODL schnell zu. Ein „Slave“ mit z.B. vier dynamisch umkonfigurierbaren „Slots“ in beliebigen Runden benötigt vier Bytes RAM für die „Shared“-Variablen. Des Weiteren belegen die „Slot“-Arbitrierungsdaten vier Bytes, die Runden-Arbitrierungsdaten zwei Bytes (pro Byte zwei Rundennummern), die „Slot“-Attribute ein Halbbyte (pro „Slot“ ein Bit) und die Rundenlängen 16 Bytes. Abhängig von den µC-Ressourcen liegen diese Daten entweder in einem gepufferten RAM oder in einem „Shadow“-EEPROM. Denn erstens soll der „Slave“ dynamisch umprogrammierbar sein und zweitens soll er nicht bei jedem Stromausfall seine RODL verlieren. Wenn man den Speichertyp einmal außer Acht lässt, so beläuft sich der RAM-Bedarf der RODL auf 27 Bytes. Die 16 „Fireworks“-Bytes können weiterhin als Konstanten im Programmspeicher (FLASH) liegen. Bei moderaten Applikationen benötigt man für diesen „Slave“-Typus etwa 2 KB FLASH und ca. 50 Bytes RAM und/oder „Shadow“-EEPROM. Plattform hierfür sind „low-cost“ µCs im Mittelfeld (≈ ..1 €).

Der „Master“ nimmt hingegen an allen Runden teil. Zum einen gestattet ihm das mehr Fehler zu erkennen (z.B. „Slave“-Ausfälle und Buskollisionen) und zum anderen ist das im „Sub“-Netzbetrieb sowieso notwendig. Denn in der Regel interessiert sich eine übergeordnete Instanz für alle Prozessdaten. Im Maximalausbau, also bei 16 Runden à 254 Daten-„Slots“, beträgt der RAM- bzw. „Shadow“-

[51] Meistens ist das SRAM und in „low-cost“ µCs so genannte „Register Files“.

[52] FLASH-Speicher kommt mittlerweile häufig vor. Trotzdem gibt es noch genug µCs mit EEPROM, EPROM und PROM.

EEPROM-Bedarf ca. 4,5 KB. Hier wurde vorausgesetzt, dass man mit einem zweidimensionalen „Array“ arbeitet. Die Runden- und „Slot“-Nummern werden nicht extra gespeichert, sondern dienen als Indizes. Ansonsten ist der Speicherbedarf noch größer. Für den „Master“ stellt das i.A. kein Problem dar, weil er als ressourcenstärkster Rechner min. über mehrere 10 KB RAM und FLASH verfügt. Prinzipiell gilt für einen universellen „Slave“ das Gleiche. Er muss zwar nicht an allen Runden teilnehmen, aber Platz dafür vorsehen. Die Plattform für diese großen Knoten sind 16 und 32 Bit-µCs mit externem Speicher (≈ 5.. €).

Darstellung

Obwohl es kein festes RODL-Format gibt, lässt sich ein generisches Format bzw. eine einheitliche Darstellung definieren. Im praktischen Teil dieser Arbeit hat sich eine Tabelle bewährt. Ihren prinzipiellen Aufbau zeigt Tabelle 3.2. In dieser Darstellungsform lässt sich die RODL eines Knotens schnell überblicken, wodurch „Scheduling“-Fehler einfacher lokalisiert werden können. Aus diesem Grund arbeitet der Projektpartner Universität Stuttgart an einem Softwarewerkzeug zur generischen RODL-Erstellung, dessen Bedieneroberfläche in etwa so aussehen wird.

Runde	Länge	„Slot“						
	[„Slots“]	0		1		2		...
0	L0	A.	V.	A.	V.	A.	V.	...
1	L1	A.	V.	A.	V.	A.	V.	...
2	L2	A.	V.	A.	V.	A.	V.	...
...	…	...	...	...	...	...	...	...
15	L15	A.	V.	A.	V.	A.	V.	...

A.: Attribut (read/write/idle)
V.: Variable

Tabelle 3.2: Bewährtes generisches RODL-Format

3.2.4 Minimalknoten

Für einen „Basic-TTP/A“-Knoten benötigt man minimal eine unterbrechbare CPU, nicht flüchtigen Speicher (FLASH), Arbeitsspeicher (RAM), einen Zeitgeber („Timer“), einen UART (Hard- oder Software)[53], einen Pegelwandler/Leitungstreiber („Transceiver“) und einen Oszillator (extern oder intern). Die prinzipielle Verschaltung dieser Komponenten zeigt Bild 3.4. Der Programmcode des Protokolls, die residenten Teile der RODL und die Applikation befinden sich im FLASH. Wie im letzten Punkt erwähnt, kommt man mit 2 KB FLASH in vielen Anwendungen zurecht. Insbesondere auf der „Slave“-Seite, da die Applikationen hier i.A. deutlich kleiner ausfallen als auf dem „Master“. Für den existierenden Programmcode von „Basic-TTP/A“ werden etwa 1,5 KB FLASH benötigt („Master“ und „Slave“). Durch Optimierung kann er etwas verkleinert werden, so dass für einen „Slave“ mit einfacher Applikation ca. 1..1,5 KB FLASH erforderlich sind. Der „Master“ kann durchaus 20 KB FLASH und mehr belegen. Denn je einfacher die „Slaves“ sind und je größer das Netz ist, desto mehr Aufgaben muss der „Master“ übernehmen. Außerdem ist bei ihm die RODL deutlich größer. Bezüglich des RAM-Bedarfs sind die Relationen ähnlich. Ein „Slave“ benötigt zwischen 10 und 100 Bytes,

[53] Bei einem gepufferten UART, wie z.B. der 16C550 im PC, muss die „Trigger“-Schwelle des Empfangs-FIFOs auf eins gesetzt werden, damit er wie ein gewöhnlicher UART reagiert.

wobei die RODL-Implementierung und die Applikation den genauen Bedarf festlegen. Beim „Master“ sollte man, allein wegen der größeren RODL, mit mehreren KB RAM rechnen.

„Timer“ und UART

Der „Timer“ ist für die zeitliche Platzierung der „Slots“ verantwortlich, weshalb er die Bezeichnung „Slot-Timer“ erhält. Je höher seine Auflösung ist, desto geringer ist der Platzierungsfehler. Auf der anderen Seite steigt mit der Auflösung die Breite des „Slot-Timers“. Wegen des Einsatzes von ressourcenarmen „low-cost“ µCs, gilt es einen Kompromiss zu finden. Wie die Berechnungen des Abschnitts 4.1 zeigen werden, reicht eine Quantisierung von ungefähr $T_{M,Slot}/100$, bei max. Rundenlänge, aus. $T_{M, Slot}$ ist die „Slot“-Dauer des „Masters“ und somit die Referenz. Für die direkte Platzierung der „Slots“ (0..254) ist ein 16 Bit-„Timer“ notwendig. Unter Zuhilfenahme eines 8 Bit-„Slot“-Zählers, genügt jedoch ein 8 Bit-„Timer“. Letzteren findet man in allen „low-cost“ µCs, einen 16 Bit-„Timer“ hingegen selten. Die „Timer“-Erweiterung mit dem „Slot“-Zähler ist keine Einschränkung. Es wird kaum zusätzlicher Speicherplatz und unwesentlich mehr Rechenzeit benötigt. Außerdem sollte eine Implementierung mit einer „Slot“-Zählung arbeiten. Nicht unbedingt für die „Slot“-Platzierung, sondern für eine redundante Kontrolle des Busgeschehens.

Im „Master“ ist der „Slot-Timer“ fest mit dem UART gekoppelt. Der hauptsächliche Grund dafür ist ein „Frame-Jitter“, den viele UARTs aufweisen (kurz UART-„Jitter“). Beim „Fireworks-Frame“ ist er besonders störend, weil die „Slaves“ für ihre Uhrensynchronisation nahezu „Jitter“-freie „Fireworks-Frames“ brauchen. Mit einem speziellen Programmkonzept und der festen Kopplung bekommt man den UART-„Jitter“ in den Griff (siehe Abschnitt 4.2). Notwendige Voraussetzung ist ein gemeinsamer Takt für den „Slot-Timer“ und den UART. In µCs mit integriertem UART ist das selbstverständlich. Externe UARTs müssen dagegen mit dem CPU-Takt angesteuert werden; aus Kostengründen ist diese Variante weniger interessant. Der „Master“ muss nicht synchronisiert werden. Schließlich ist er die Zeitreferenz. Damit er dieser Aufgabe gerecht wird, benötigt er allerdings einen genauen, temperaturstabilen Oszillator. Standardquarzoszillatoren beispielsweise reichen vollends aus.

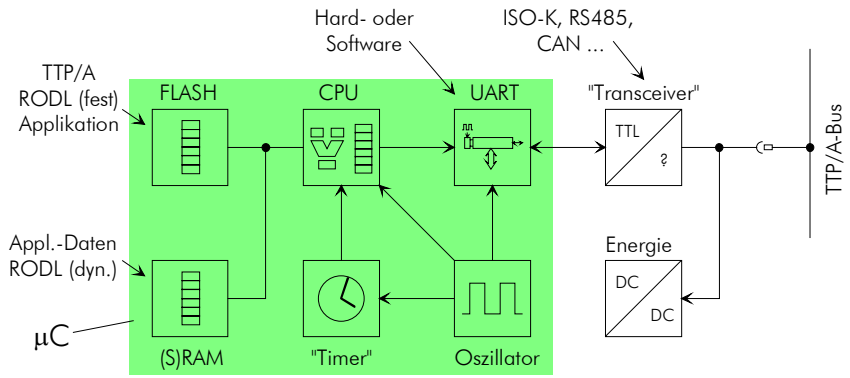


Bild 3.4: Blockschaltbild eines „Basic-TTP/A“-Minimalknotens

Bei einem „Slave“ laufen UART und „Slot-Timer“ lose gekoppelt. Für die Kommunikation darf der Bittakt eines „Slaves“ bis zu ±1,84 % von der „Master“-Bitfrequenz abweichen. Die Busarbitrierung muss jedoch wesentlich genauer sein, wie das folgende Beispiel zeigt. Eine zweiprozentige „Slot-Timer“-Abweichung würde bei maximaler Rundenlänge zu einem Versatz von $0{,}02 \cdot 253 \cdot T_{Slot} \approx 5 \cdot T_{Slot}$ führen – ein „Slave“ muss max. 253 „Slots“ vorhersagen, da der „Fireworks-Slot“ nicht mitgezählt

wird. Die Folge dieser ca. fünf „Slots“ großen Abweichung wäre eine Buskollision[54]. In Kapitel 4 wird gezeigt, dass das „Slot-Timing“ etwa hundertmal genauer sein muss als das UART-„Timing“. Theoretisch könnte man den UART fest mit dem „Slot-Timer“ verkoppeln; Letzterer ist ja viel genauer. Praktisch scheitert das jedoch an der Einstellbarkeit der Baudrate. Im Gegensatz zu „Timern“, besitzen UARTs eine viel gröbere Granularität[55]. Dadurch lässt sich die rein rechnerische Baudrate für eine feste Kopplung nur in etwa einstellen. Aus diesem Grund wird das Idealverhältnis zwischen „Slot“- und Bitdauer von 13:1 so gut wie nie entstehen; eine Ausnahme bilden homogene Netze mit quarzoszillatorbestückten Knoten. In der Regel korrelieren also Bittakt und „Slot“-Dauer in einem „Slave“ nur, weshalb der „Timer“ und der UART in einem „Slave“ auch von unterschiedlichen Oszillatoren getaktet werden dürfen.

Andere Komponenten

Zwischen dem UART und dem Bus befindet sich ein Pegelwandler mit Leitungstreiber („Transceiver“). In Bild 3.4 ist auf der Busseite ein Fragezeichen eingetragen, weil „Basic-TTP/A“ keinen festen „Physical-Layer“ vorsieht. Erlaubt sind alle standardisierten, preisgünstigen „Physical-Layer“. Bisher wurden definiert: ISO 9141 für „low-cost“ Anwendungen und RS485 bzw. CAN für Anwendungen mit höherer Störfestigkeit und Übertragungsraten bis ≈ 1 MBit/s (siehe Abschnitt 3.3). Eine weitere Komponente betrifft die Energieversorgung. In den meisten Fällen ist das ein Standardspannungsregler, linear oder getaktet. Für einen Knoten ist die Energiequelle der Bus. Momentan geschieht das über separate Leitungen, eine „Powerline“ ist jedoch angedacht.

Die meisten µCs besitzen die Komponenten in dem hervorgehobenen Bereich in Bild 3.4. Eine Liste geeigneter Bausteine befindet sich im Anhang unter Punkt 8.9. Hie und da muss man beim UART mit einer Emulation nachhelfen; Details hierzu sind in Abschnitt 8.3 nachzulesen. Für einen minimalen „Basic-TTP/A“-Knoten benötigt man nur wenige kostengünstige Standardbauteile. Aufgrund der technologischen Fortschritte wird diese Anzahl weiter sinken und mit ihr die Kosten für einen Knoten. Zukünftige µCs werden wahrscheinlich einen integrierten „Transceiver“ besitzen. Die Firma Maxim bietet z.B. schon einen UART mit RS485-„Transceiver“ an (MAX 3140). Dadurch wurde gezeigt, dass diese unterschiedlichen Technologien integrierbar und vor allem in Serie herstellbar sind.

3.2.5 Knoteninteraktionen

An einem kleinen Beispiel soll nun gezeigt werden wie die wichtigsten Komponenten eines Knotens interagieren. Im Speziellen sind das der „Slot-Timer“ und UART im „Master“ und „Slave“. Die anderen Komponenten sind zwar nicht unwichtig, für die generellen „Basic-TTP/A“-Abläufe aber nebensächlich. In dem Beispiel sollen ein „Master“ und „Slave“ vernetzt sein. Des Weiteren wird angenommen, dass nur eine Runde existiert. Die Rundennummer und -länge sind ohne Belang. Wie in Bild 3.5 dargestellt, sendet der „Slave“ in „Slot“ m und ansonsten der „Master“.

„Master“

Für den „Master“ beginnt eine Runde mit einem „Interrupt“ von seinem „Slot-Timer“. Der „Interrupt“ aktiviert eine „Task“ mit der „Basic-TTP/A“-Rundensteuerung. Da es sich um den Beginn einer Runde handelt, lädt die „Task“ den entsprechenden „Schedule“ und beschreibt das „Transmit“-Register des UARTs mit dem „Fireworks“-Byte der initiierten Runde. Kurz danach beginnt der UART zu

[54] Dieses Beispiel macht zudem deutlich, dass eine Synchronisierung von „Slaves“ mit ungenauen, temperaturempfindlichen RC- und „On-Chip“-Oszillatoren absolut notwenig ist.

[55] Das gilt für Hardware-UARTs. Software-UARTs und andere Emulationen sind i.A. feiner einstellbar.

senden. Bei den Daten-„Slots“ ist der Ablauf ähnlich. Jeweils am „Slot“-Anfang stößt der „Slot-Timer“ die Rundensteuerungs-„Task“ an. Daraufhin sieht die „Task“ im „Schedule“ nach, was in dem entsprechenden „Slot“ zu tun ist. Gemäß Bild 3.5, veranlasst sie den UART bis „Slot“ m zu senden. In „Slot“ m merkt die „Task“, dass sie zuhören muss. Es würde wenig Sinn machen die „Task“ auf den „Receive-Interrupt“ des UARTs warten zu lassen, weil dieser erst gegen Ende des „Slots“ auftritt. Hierfür ist die Rechenzeit einfach zu schade, insbesondere bei „low-cost“ µCs. Deshalb liegt es nahe, für den Empfang eine andere „Task“ einzurichten, die vom „Receive-Interrupt“ angestoßen wird. Nach „Slot“ m sendet der „Master“ wieder, und zwar bis zum Ende der Runde. Mit dem ersten „Slot-Timer-Interrupt“ nach der IRG beginnt das Ganze dann wieder von vorne. Außerdem erkennt man aus Bild 3.5, dass der „Slot-Timer“ im „Master“ in einem „Auto-Reload“-Modus arbeiten kann. Das gilt nicht nur für dieses Beispiel, weil der „Master“ generell an allen „Slots“ teilhaben sollte. Erstens, um die Fehler aller „Slaves“ registrieren zu können und zweitens, um bei einer „Gateway“-Erweiterung alle Prozessdaten parat zu haben.

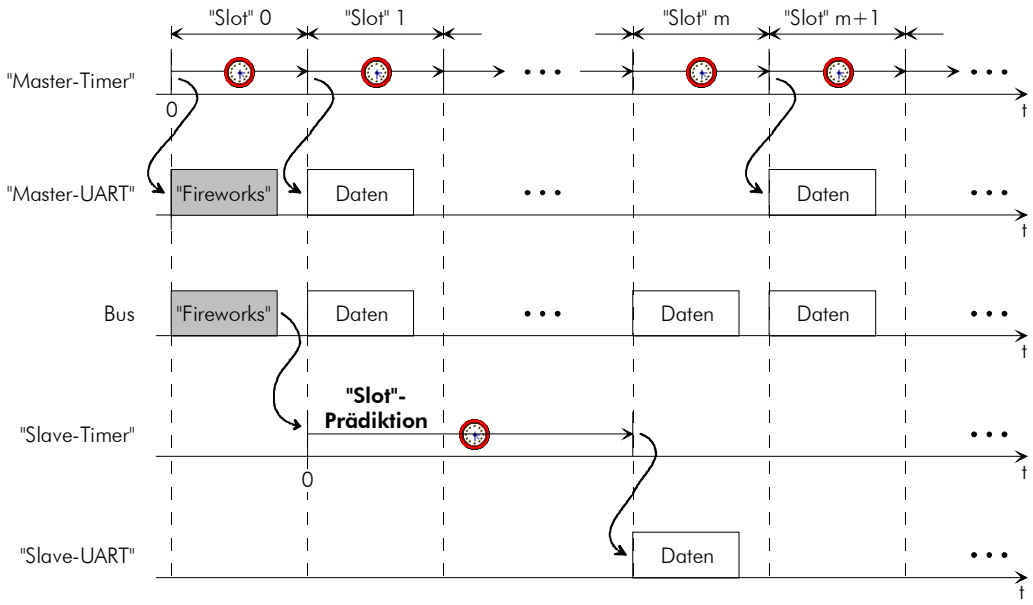


Bild 3.5: Interaktionen in einem „Basic-TTP/A-Master“ und „-Slave“

„Slave“

Bei einem „Slave“ ist das anders. Er nimmt typischerweise nur an ein paar „Slots“ einer Runde teil; eine Ausnahme ist z.B. ein Universal-„Slave“ für Testzwecke. Des Weiteren unterscheidet sich der Rundenstart bei einem „Slave“. Für ihn beginnt eine Runde nach dem „Fireworks-Frame“ (siehe „Slot-Timer“-Nullpunkte in Bild 3.5). Vorher weiß er ja nicht, welche Runde der „Master“ initiieren wird. Nach Erhalt des „Fireworks-Frames“ führt der „Slave“ eine Fehlerprüfung durch. Fällt sie positiv aus, so verwirft er den angeblichen „Fireworks-Frame“ und wartet auf den Nächsten. Eine Vorwärtskorrektur macht hier wenig Sinn, da bei einem Irrtum die Kommunikation der anderen Knoten eine ganze Runde lang gestört werden kann. Ist das Ergebnis der Fehlerprüfung negativ, dann lädt der „Slave“ den „Schedule“ für diese Runde. Anschließend korrigiert er die kumulierten Zeitfehler der letzten Runde, indem er die Phase seines „Slot-Timers“, so wie in Bild 3.5 dargestellt, auf den Beginn des ersten Daten-„Slots“ synchronisiert. Der „Slave“ benutzt dazu den „Receive-Interrupt“ seines

UARTs, der nach dem „Fireworks-Frame“ auftritt[56]. Ein anderer „Frame“ eignet sich hierfür nicht, weil man nur beim „Fireworks-Frame“ sicherstellen kann, dass er vom „Master“ (Zeitreferenz) stammt. Nach dem „Fireworks-Slot“ beginnt die Prädiktion für „Slot“ m. Um Rechenzeit zu sparen, zieht der „Slave“ seinen „Slot-Timer“ mit dem Wert $(m - 1)\cdot T_{Slot}$ auf. Nach Ablauf dieser Zeit startet der „Slot-Timer“ eine „Task“, die den UART zum Senden des entsprechenden Datums veranlasst. Für das Empfangen von Daten ist deutlich weniger Aufwand nötig, weil keine Buskonflikte entstehen können. Grob gesprochen muss der „Slave“ am Bus lediglich mithören und die für ihn bestimmten „Frames“ herausfiltern. Nach dem Ende einer Runde wartet der „Slave“ auf den nächsten „Fireworks-Frame“, um den beschriebenen Vorgang erneut zu starten.

Realzeitrestriktionen

Ganz so einfach sind die Vorgänge allerdings nicht. Sie wurden hier bewusst einfach geschildert, damit die prinzipiellen Zusammenhänge klar werden. In der Praxis gibt es aber jede Menge Schmutzeffekte, die hauptsächlich bei der Realzeitthematik ernsthafte Probleme machen, insbesondere bei höheren Baudraten. Ein Beispiel ist der so genannte UART-„Jitter“. Wenn die „Fireworks-Frames“ hin und her wackeln, können sie die „Slaves“ nicht als Referenzmarken verwenden. Das gilt für die angerissene Phasensynchronisation[57] und für die noch nicht näher beschriebene Frequenzsynchronisation. Letztere ist nur bei ungenauen Oszillatoren (RC oder „On-Chip“) notwendig. Die Phase muss immer korrigiert werden, auch wenn alle Knoten sehr gute Oszillatoren besitzen. Es wäre nur eine Frage der Zeit, bis ein freilaufender „Slave“ einen Buskonflikt verursachen würde. Der Sachverhalt ist so ähnlich wie bei einem UART. Dort stimmen die Bitfrequenzen in etwa überein, so dass die Empfänger eine Zeit lang selbstständig arbeiten können. Sie müssen aber an jeder Startbitflanke ihre Uhren neu justieren, weil sonst der toleranzbedingte, kumulierte Zeitfehler zu groß würde. Die Folgen einer versetzten Startbitflanke wären engere Toleranzen für die Bitfrequenzen und im Extremfall falsche Bitabtastungen. Weitere Schmutzeffekte sind Latenzzeiten, Programmausführungszeiten, „Critical-Sections“ usw. Darüber, und über eine erfolgreiche Lösungsstrategie, wird in Kapitel 4 ausführlich berichtet.

3.2.6 Verbindungsplanung

Kommunikationsverbindungen (kurz Verbindungen) werden durch die „Schedules“ definiert. „Basic-TTP/A“ ist hier flexibel. Für jeden „Slot“ kann entweder eine „Broadcast“-, „Multicast“- oder „Point-to-Point“-Verbindung realisiert werden. Weil das „Scheduling“ Teil der Applikation ist, entstehen damit nützliche Freiheitsgrade. „Broadcasts“ ermöglichen systemweite Nachrichten oder Daten. Eine typische Anwendung ist die Notausfunktion einer Maschine oder Produktionsanlage. „Multicasts“ sind einschränkte „Broadcasts“. Mit ihnen kann man z.B. eine Knotengruppe synchronisieren oder einen „Slave“ zu einem „Basic-TTP/A“-Monitor umfunktionieren. Und schließlich die „Point-to-Point“-Verbindungen. Bei ihnen unterscheidet man zwischen direkt und indirekt. Eine indirekte „Point-to-Point“-Verbindung ist die klassische „Master/Slave“-Kommunikation. Sie wird hauptsächlich bei „Slaves“ eingesetzt, deren Ressourcen nicht zur Verarbeitung der Daten ausreichen. Diese Aufgabe übernimmt der besser ausgestattete „Master“. Außerdem kann man indirekte „Point-to-Point“-Verbindungen für Manipulationen verwenden, z.B. bei einer Fernsteuerung. Es ist natürlich möglich die Rechenlast auf mehrere Knoten zu verteilen. Im Gegensatz dazu transportiert eine direkte „Point-to-Point“-Verbindung Daten ohne Umwege vom Sender zum Empfänger. Der Vorteil ist dabei eine kleinere Latenzzeit, was besonders für Regelkreise wichtig ist, die über den Bus geschlossen sind. Jede

[56] Die Art der Kommunikation (synchron oder asynchron) ist prinzipiell nebensächlich.

[57] Eigentlich ist der Ausdruck nicht ganz korrekt, weil die Frequenzen nicht exakt gleich sind.

Latenzzeit vergrößert die Totzeit, wodurch der Stabilitätsbereich eingeschränkt wird. Allerdings benötigt man für direkte „Point-to-Point“-Verbindungen „Slaves“ mit ausreichenden Ressourcen, da sie die Prozessalgorithmen selbst ausführen müssen.

Die Umsetzung soll an dem kleinen Beispiel aus Bild 3.6 demonstriert werden. Es ist ein „Stand-Alone-Basic-TTP/A“-Netz mit einem „Master“ und je zwei Sensor- und Aktor-„Slaves“ zu sehen. Für die Kommunikation soll eine Runde mit sechs „Slots“ genügen. Neben dem „Basic-TTP/A“-Netz befinden sich die „Schedules“ für die Knoten. Sende-„Frames“ sind mit einem „w“ (write) gekennzeichnet und grau unterlegt. Lese-„Frames“ enthalten dagegen nur ein „r“ (read). Der Datensemantik wird keine Beachtung geschenkt, weil hier das „Scheduling“ im Vordergrund steht.

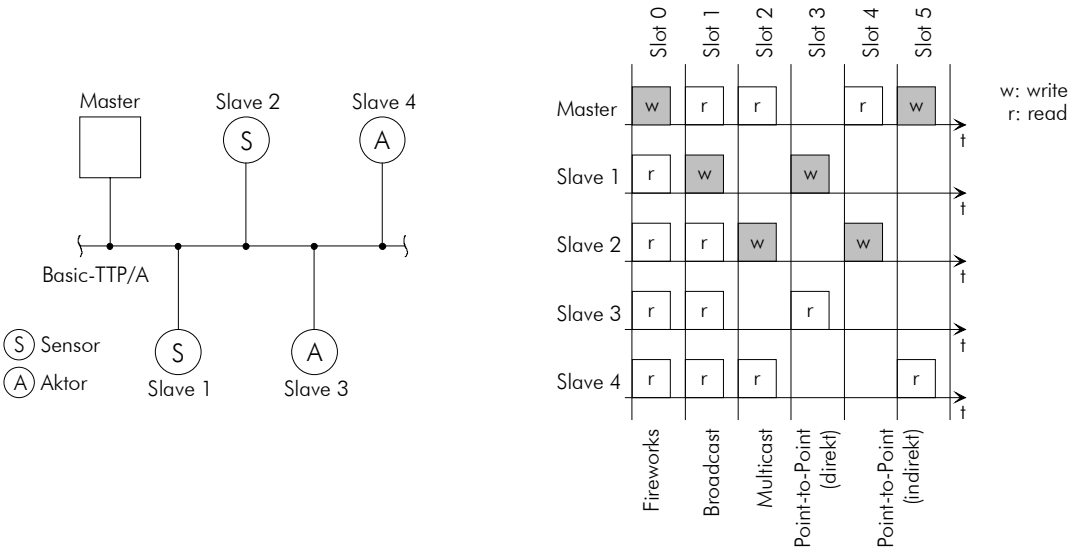


Bild 3.6: Einrichten von Verbindungen

Wie üblich, beginnt die Runde mit dem „Fireworks-Slot“. Das ist ein „Master-Broadcast“ in „Slot“ 0, auf den man, mit Ausnahme der Rundennummer, keinen Einfluss hat. „Slot“ 1 enthält einen weiteren „Broadcast“, der jedoch über das Anwendungs-„Scheduling“ definiert ist. In diesem „Slot“ soll „Slave“ 1 seinen Sensorwert an alle anderen Knoten verschicken. Dementsprechend muss die Applikation in „Slave“ 1 „Slot“ 1 als „Write-Slot“ einrichten. Hingegen müssen die Applikationen in den restlichen Knoten für diesen „Slot“ das Attribut „read“ festlegen. Für „Slot“ 2 ist ein „Multicast“ mit „Slave“ 2 (Sensor) als Datenquelle definiert. Das Datum ist für den „Master“ und „Slave“ 4 bestimmt. Alle anderen „Slaves“ ignorieren diesen „Slot“. Analog zum vorherigen „Slot“, richten die Applikationen in „Slave“ 2, „Slave“ 4 und im „Master“ den „Slot“ 2 als „Write-„ bzw. „Read-Slot“ ein. Die Applikationen in „Slave“ 1 und „Slave“ 3 tragen dagegen als „Slot“ 2-Attribut „idle“ ein. In den nächsten „Slots“ sind „Point-to-Point“-Verbindungen realisiert. „Slot“ 3 zeigt eine direkte Kommunikation zwischen „Slave“ 1 und „Slave“ 3 und „Slot“ 4 und 5 den „Master/Slave“-Klassiker. Weitere, angewandte Beispiele enthält Kapitel 5.

3.2.7 Daten-„Frame“

In Daten-„Frames” transportiert „Basic-TTP/A“ beliebige Applikationsdaten byteweise. Die Applikationsdaten können Prozess-, Konfigurationsdaten, Nachrichten oder Befehle sein. „Basic-TTP/A“ unterscheidet hier nicht. Für die Interpretation ist entweder die Applikation oder einer der anderen

TTP/A-Bausteine verantwortlich. Ein Daten-„Frame“ ist ein Standard-UART-„Frame“ mit dem Format: Acht Datenbits, ungerades („odd“) Partitybit und ein Stoppbit. Inklusive Startbit beträgt die Länge des Daten-„Frames“ elf Bits, so wie in Bild 3.7 dargestellt. Daten mit mehr als acht Bits müssen durch die Applikation auf mehrere Daten-„Frames“ verteilt werden. Bei den typischerweise kleinen Datenpaketen im Sensor/Aktorbereich, kommt das jedoch selten vor. Das Paritybit erfüllt zwei Aufgaben. Es unterscheidet zum einen Daten- und „Fireworks-Frames“ (inverse Parität) und dient zum anderen der Fehlersicherung. Mit dem Paritybit werden alle ungeradzahligen Bitfehlermuster (1, 3, 5, 7, 9) sicher erkannt. Die min. Codedistanz beträgt jedoch nur $d_{min} = 2$. Entsprechend gering fällt der Fehleraufdeckungsgrad mit $1 - 2^{-1} = 0{,}5$ aus. Von größerer Bedeutung ist jedoch die Restfehlerwahrscheinlichkeit. In Abschnitt 4.3.3 wird gezeigt, dass sie für eine mittlere Bitfehlerwahrscheinlichkeit von 10^{-3} ungefähr bei $3{,}6 \cdot 10^{-5}$ liegt. Damit erfüllen idempotente Daten, was für „Basic-TTP/A“ typisch ist, die Kriterien der Datenintegritätsklasse I1 nach DIN 19244 (siehe Punkt 8.7). Für einfache, kostengünstige Anwendungen reicht das vollends aus. Sicherheitskritische Anwendungen stellen allerdings höhere Ansprüche[58]. Hierfür benötigt man zusätzlich den TTP/A-Baustein „Extended-Reliability“ (siehe Kapitel 6).

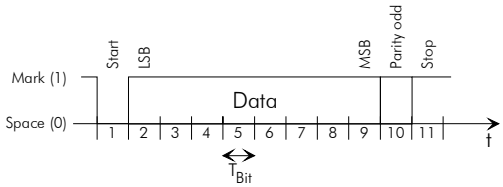


Bild 3.7: Aufbau eines Daten-„Frames“

3.2.8 „Fireworks-Frame“

Rein äußerlich unterscheidet sich ein „Fireworks-Frame“ kaum von einem Daten-„Frame“. Er ist ebenso ein elf Bit langer Standard-UART-„Frame“ mit einem Startbit, acht Datenbits, einem Paritybit und einem Stoppbit. Das Paritybit ist jedoch gerade („even“), um die Anzahl der Unterscheidungsmerkmale gegenüber einem Daten-„Frame“ zu erhöhen. Die Redundanz entsteht aufgrund der Position eines „Fireworks-Frames“. Er ist der erste „Frame“ in einer Runde („Slot“ 0) und normalerweise der Einzige, dem eine längere Pause (IRG) vorangeht. Den Aufbau eines „Fireworks-Frames“ zeigt Bild 3.8.

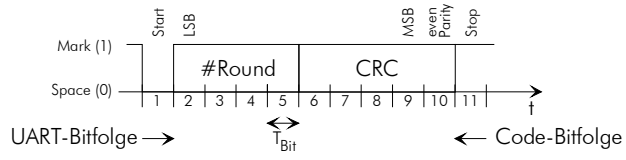


Bild 3.8: Aufbau eines „Fireworks-Frames“

Die vorderen vier Bits (LSBs) enthalten die Rundennummer (0..15) und die hinteren Bits (MSBs) eine CRC-Prüfsumme. „Fireworks-Frames“ sind also deutlich besser gegen Übertragungsfehler gesichert als Daten-„Frames“. Der Grund dafür ist, dass ein unerkannter Fehler in einem „Fireworks-Frame“

[58] Der Einsatz von Lichtwellenleitern, mit einer typischen Bitfehlerwahrscheinlichkeit von 10^{-12} [100], ändert daran meistens nichts, weil auch die Vorgaben für die Restfehlerwahrscheinlichkeit sinken.

i.A. weitaus schädlicher ist als in einem Daten-„Frame“. Ein nicht identifizierter, byzantinischer Fehler ist der „Worst-Case“. Hier empfängt mindestens ein „Slave“ ein verfälschtes, aber gültiges „Fireworks-Frame“. Weil der „Worst-Case“ angenommen wird, soll die Rundennummer in Gebrauch sein – es werden eher selten alle 16 Rundennummern verwendet. Demzufolge arbeitet mindestens ein „Slave“ mit einem falschen „Schedule“, wodurch es erstens zu einer Vertauschung der Datensemantik kommt und zweitens zahlreiche Buskonflikte auftreten können. Dazu muss dem betroffenen „Slave“ lediglich eine Runde vorgetäuscht werden, in der er oft sendet. Wenn diese Runde zudem sehr lang ist und in der IRG zufällig Ruhe herrscht, können nachfolgende Runde in Mitleidenschaft gezogen werden. Im Gegensatz dazu sind von einem unerkannten, verfälschten Datum meistens nur wenige Knoten kurzzeitig betroffen. Das falsche Datum wird spätestens mit der nächsten Aktualisierung korrigiert. Außerdem kann ein verfälschtes Datum keinen Buskonflikt bewirken. Auf die Uhrensynchronisation wirken sich Übertragungsfehler nicht so dramatisch aus. Dort sind zum einen die Rundennummern nicht so wichtig. Zum anderen weiß man, dass der Prozess stetig verläuft, weshalb algorithmische Korrekturen möglich sind.

CRC-Sicherung

Die Entscheidung für das CRC-Verfahren zur Fehlersicherung hat mehrere Gründe. Es basiert auf einer ausgereiften Theorie und hat sich in der Praxis vielfach bewährt [104] [109] [108] [110]. Die Stärken der CRC-Codierung liegen in der Erkennung und weniger in der Vorwärtskorrektur von Übertragungsfehlern. Da Letzteres für die „Fireworks-Frames“ nicht in Frage kommt, passt diese Stärke gut zu den Anforderungen der TTP/A-Fehlersicherung. Ein weiterer Punkt ist die Erkennung von „Burst“-Fehlern[59] über die Hamming-Distanz d_{min} hinaus. Mit CRC-Codes lassen sich „Burst“-Fehler der Länge m sicher erkennen, wobei m der Grad des Generatorpolynoms ist. Längere „Burst“-Fehler werden mit einer hohen Wahrscheinlichkeit aufgedeckt. Die Hamming-Distanz d_{min} ist in der Regel kleiner als m – allgemein weiß man bei CRC-Codes nur, dass $d_{min} \geq 4$ ist [104]. Für „Basic-TTP/A“ spielen „Burst“-Fehler insofern eine Rolle, da das Bussystem aus Kostengründen hauptsächlich von der drahtgebundenen Übertragung Gebrauch macht. „Burst“-Fehler sind in diesen Kommunikationskanälen zwar kein Muss, sie treten aber des Öfteren auf. Ausschlagend ist das EMV-Umfeld, weshalb keine generelle Aussage möglich ist. Typische Beispiele mit „Burst“-Störquellen sind Kfzs, Schweißroboter und Servoantriebe.

Des Weiteren lässt sich das CRC-Verfahren effizient in einem µC implementieren, insbesondere wenn der Coderaum klein ist – ein zusätzlicher Hardware-Codierer/Decodierer ist für „Basic-TTP/A“ zu teuer. Man berechnet die 16 Codewörter der „Fireworks-Frames“ a priori und legt sie in einer Tabelle ab. Dafür benötigt man lediglich 16 Bytes (FLASH), womit die Implementierbarkeit auf „low-cost“ µCs gewährleistet ist. Außerdem codiert die CRC-Methode systematisch, wodurch sich die Rundennummern unverschlüsselt in den Codewörtern befinden (siehe Bild 3.8). Für die Codierung und Decodierung ist das vorteilhaft, weil man die Rundennummern als Tabellenindizes verwenden kann. Das Resultat dieser direkten Methode ist ein kleiner und schneller Algorithmus, der auch bei 625 kBit/s, auf einem nur 10 MIPS starken µC, problemlos arbeitet (siehe Kapitel 5).

Für die Absicherung der Rundennummern wurde eigens das Generatorpolynom in Formel 3.1 entwickelt. Alle bekannten Generatorpolynome [109] erzeugen zu lange Codes. Dies ist aus folgenden Gründen ungünstig: Erstens leidet die Übertragungseffizienz darunter, weil mindestens zwei „Frames“ benötigt werden. Zweitens verändert die Trennung der Codewörter die Eigenschaft der „Burst“-Fehlererkennung, wodurch der Einsatz eines CRC-Codes fragwürdig wird. Drittens sinkt mit jedem Bit

[59] „Burst“-Fehler beginnen und enden mit einem verfälschten Bit. Dazwischen dürfen sich auch unverfälschte Bits befinden. Deshalb sieht das Fehlermuster wie folgt aus: 1xxx..1.

die Wahrscheinlichkeit für eine fehlerlose Übertragung. Und viertens benötigt man für eine ausreichende Absicherung der Rundennummern nicht mehr als fünf Prüfbits. Eine Rundennummer besteht aus vier Bits, weshalb sich mit wenig Redundanz eine gute Hamming-Distanz erzielen lässt. Wie sich zeigen wird, reicht das für sichere oder verlässliche Systeme nicht aus, aber für einfache, preisgünstige Anwendungen und das ist der Fokus von „Basic-TTP/A“. Anspruchsvollere Anwendungen benötigen den optionalen TTP/A-Baustein „Extended-Reliability“ (siehe Abschnitt 3.4 und Kapitel 6).

$$g(x) = (x^4 + x^3 + 1) \cdot (x + 1) = x^5 + x^3 + x + 1$$

Formel 3.1: CRC-Generatorpolynom für den „Fireworks“-Code

Bei der Entwicklung der so genannten „Fireworks“-Codes galt es zwei Hürden zu nehmen. Die Erste war die Codewortlänge. Bild 3.8 zeigt, dass sie neun Bits beträgt. In einem UART-„Frame“ stehen allerdings nur acht Bits, die Informationsbits, frei zur Verfügung. Auf das neunte Bit, das Paritybit, hat man i.A. keinen Einfluss, da es der UART berechnet[60]. Ein Stück weiter oben wurde erwähnt, dass CRC-Codes eine Hamming-Distanz $d_{min} \geq 4$ aufweisen müssen. Mit acht Bits kann man jedoch nur (8,4)-Codes[61] konstruieren, deren Hamming-Distanz max. drei beträgt. Eine Hamming-Distanz $d_{min} = 4$ erreicht man erst mit ausgewählten (9,4)-Codes. Aus diesem Grund, und natürlich wegen der kleineren Restfehlerwahrscheinlichkeit, ist das Generatorpolynom in Formel 3.1 so konstruiert worden, dass es einen (9,4)-CRC-Code mit $d_{min} = 4$ und Paritybit erzeugt[62]. Für das Paritybit ist der Faktor (x + 1) verantwortlich. Der vordere Faktor ist ein Primitivpolynom vierter Ordnung. Davon gibt es zwei Stück [109]. Beide liefern gleich gute (9,4)-CRC-Codes, weshalb der Zufall als Entscheidungshilfe herangezogen wurde. Die Daten der „Fireworks“-Codierung lauten:

- minimale Codedistanz $d_{min} = 4$
- Fehleraufdeckungsgrad $1 - 2^{-5} = 0{,}96875$
- alle ungeraden Bitfehlermuster (1, 3, 5, 7, 9) werden erkannt
- „Burst“-Fehler bis zu einer Länge von fünf Bits werden erkannt; in diesem Fall ist das für „Burst“-Fehler der Länge vier wichtig, weil alle ungeraden Bitfehler sowieso erkannt werden
- „Burst“-Fehler der Länge 6 werden mit einer Wahrscheinlichkeit von $1 - 2^{-4} = 0{,}9375$ erkannt
- „Burst“-Fehler größer 6 werden mit einer Wahrscheinlichkeit von $1\text{-}2^{-5} = 0{,}96875$ erkannt
- für eine mittlere Bitfehlerwahrscheinlichkeit von 10^{-3} beträgt die Restfehlerwahrscheinlichkeit ungefähr $3{,}91 \cdot 10^{-12}$ (siehe Abschnitt 4.3.3); Datenintegritätsklasse I2 nach DIN 19244 (siehe Punkt 8.7); insgesamt aber Datenintegritätsklasse I1, wegen den Daten-„Frames“

Die Anordnung der Bits stellt die zweite Hürde dar. Bei einem CRC-Code befinden sich die Prüfstellen in den niederwertigen Bits, da ein Codewort immer ein Vielfaches des Generatorpolynoms ist[63]. Folglich ist der Platz des Codewort-Paritybits im LSB. Gemäß Bild 3.8, ist das Paritybit in einem UART-„Frame“ jedoch das MSB – damit ist nicht das MSB des Informationsbytes gemeint, es wird lediglich die Regel „LSB first“ konsequent weitergeführt. Deshalb müssen die Codewörter so umge-

[60] Es gibt nur wenige UARTs, die das Paritybit als zusätzliches Informationsbit nutzen können.

[61] Die Bezeichnung (n,k)-Code besagt, dass der Code n Stellen umfasst, von den k die Information betreffen.

[62] Diese Berechnungen sind aufwändig und wurden mit einem selbstentwickelten Programm rechnergestützt durchgeführt.

[63] $c(x) = i(x) \cdot x^m - r(x) = q(x) \cdot g(x)$ mit i(x) Informations-, g(x) Generator-, r(x) Rest- und q(x) Teilerpolynom.

stellt werden, dass beide Paritybits zusammenfallen. Sonsten kann man die neun Bits im UART-„Frame“ nicht nutzen. Bei der Bitumstellung ist Vorsicht geboten. Permutieren der Bits wirkt sich zwar nicht auf die min. Codedistanz aus, unter Umständen aber auf die „Burst“-Fehlererkennung. Glücklicherweise verhält sich eine Spiegelung der Bits (MSB ↔ LSB usw.) neutral und liefert die gewünschte Vertauschung. Bei dieser Operation wird ein „Burst“-Fehler nicht zerlegt, wodurch er nach wie vor erkennbar ist. Allerdings ändern sich die Rundennummern. Durch das Spiegeln der Bits tauschen die Rundennummern 2 und 4, 1 und 8, 3 und 12 etc. ihre Plätze. Bis auf eine Neusortierung hat das keine Auswirkung.

Das Ergebnis ist in Tabelle 3.3 zu sehen. Spalte 1 enthält die Rundennummern und Spalte 2 die Bitsequenzen in den UART-„Frames“ – das sind die CRC-Codewörter. In Spalte 3 befinden sich die so genannten „Fireworks“-Bytes. Sie sind die Informationsbytes in den UART-„Frames“, denn das Paritybit berechnet oder wertet ein UART selbstständig aus. Als Ergänzung listet die letzte Spalte die Rundennummern vor der Spiegelung auf. Die Codierung geschieht durch das Beschreiben des UARTs mit dem gewünschten „Fireworks“-Bytes. Hingegen muss man für die Decodierung das empfangene „Fireworks“-Byte mit dem entsprechenden Tabelleneintrag vergleichen und das „Parity-Error-Flag“ des UARTs auswerten. Nur wenn beide Tests keine Fehler anzeigen, wird die übertragene Rundennummer als gültig angesehen.

Rundennummer dez.	**UART-Sequenz bin.**			**„Fireworks“-Byte hex.**	**Spiegelnummer dez.**
	L	M	P		
0	0000	0000	0	0x00	0
1	0001	0101	1	0xA8	8
2	0010	1011	0	0xD4	4
3	0011	1110	1	0x7C	12
4	0100	0011	1	0xC2	2
5	0101	0110	0	0x6A	10
6	0110	1000	1	0x16	6
7	0111	1101	0	0xBE	14
8	1000	0111	0	0xE1	1
9	1001	0010	1	0x49	9
10	1010	1100	0	0x35	5
11	1011	1001	1	0x9D	13
12	1100	0100	1	0x23	3
13	1101	0001	0	0x8B	11
14	1110	1111	1	0xF7	7
15	1111	1010	0	0x5F	15

Tabelle 3.3: „Fireworks“-Codes

3.2.9 „Interround-Gap“

Die „Interround-Gap“ (IRG) dauert $13 \cdot T_{Bit}$ und ist genauso lang wie ein „Slot“. Sie ist eine Pause zwischen zwei Runden, sorgt für deren klare Trennung (redundantes Merkmal für ein „Fireworks-Frame“)

und hilft startenden „Slaves“ bei der Synchronisation. Ein Synchronisationsproblem kann beim Einschalten oder „Hot-Plug-In“ entstehen. Angenommen ein „Slave“ hat die richtige Baudrate eingestellt und beginnt gerade am Bus mitzuhören. Sein UART wird die nächste fallende Flanke als Startbitflanke interpretieren. Ob es sich tatsächlich um eine Startbitflanke handelt, ist nicht sicher. Es kann genauso gut eine beliebige „10“-Bitfolge in einer Runde sein. In einem kontinuierlichen Bitstrom ist es sogar möglich, dass ein aufsynchronisierender „Slave“ nie eine Startbitflanke findet. Aus diesem Grund sind wiederkehrende Unterbrechungen mittels den IRGs notwendig. Bei einer angemessenen Dauer garantieren sie die Synchronität der UARTs in den Knoten und damit die Kommunikation.

Anhand von Bild 3.9 soll die Bestimmung der Minimaldauer erfolgen. Im ungünstigsten Fall kann ein „Slave“ außer Tritt das letzte Paritybit einer Runde fälschlicherweise als Startbit ansehen. Sein UART tastet dann die nächsten elf Bitzeiten ab. Erst im Anschluss ist er wieder für den Empfang eines Startbits bereit. Aus Bild 3.9 kann man ablesen, dass dieser Zeitpunkt sieben Bitzeiten nach dem Rundenende liegt. Hierbei wurden jedoch stillschweigend exakt platzierte „Frames“ und keine Baudratentoleranzen vorausgesetzt. Unter Beachtung dieser Schmutzeffekte kann ein UART auch später leer laufen, ca. 9,5 Bitzeiten nach Rundenende[64]. Die Schmutzeffekte sind jedoch nebensächlich, da in der 13 Bitzeiten langen IRG genug Spielraum vorhanden ist. Der Grund für die Überdimensionierung liegt in der Berechnung von Zeitintervallen. Sie ist wesentlich einfacher, wenn die IRG genauso lang wie ein „Slot“ ist, da nur eine atomare Zeiteinheit existiert. Die Auswirkungen auf die Übertragungseffizienz sind vernachlässigbar.

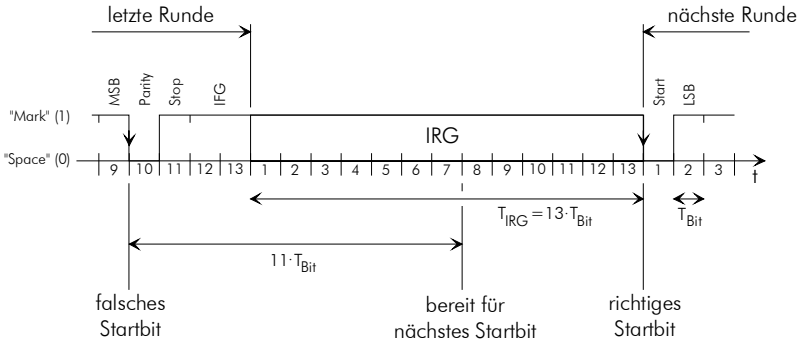


Bild 3.9: „Worst-Case“-Betrachtung für die IRG

3.2.10 „Interframe-Gap“

So wie die IRG zwei Runden voneinander trennt, so trennt die „Interframe-Gap“ (IFG) zwei „Frames“ voneinander. Ihre Funktion ist jedoch gänzlich verschieden. Mit der IFG werden Buskollisionen benachbarter „Frames“ aufgrund von unvermeidlichen Zeitfehlern verhindert. Ihren Ursprung haben die Zeitfehler in den verteilten Uhren, Latenz- und Laufzeiten. Jeder Knoten besitzt eine unabhängige Uhr für die zeitliche Steuerung der „Basic-TTP/A“-Vorgänge. Durch nicht perfekte Oszillatorabstimmungen, Temperaturdrifts, Alterungseffekte und verschiedene Granularitäten, werden die Uhren in den Knoten nie exakt die gleiche Zeit anzeigen. Außerdem verhindern Signallaufzeiten, dass ein „Frame“ alle Knoten zur selben Zeit erreicht. „Jitter“, „Critical-Sections“, „Interrupt-Latenzen“ und einiges mehr sorgen für zusätzliche Unsicherheiten. Mit entsprechenden Synchronisationsmechanismen können diese Fehler kompensiert werden. Allerdings nicht vollständig, weshalb ein Restfehler bleibt, den

[64] „Frame“-Versatz bis etwa $2 \cdot T_{Bit}$ und „Frame“-Streckung bis etwa $0{,}5 \cdot T_{Bit}$.

man mit der IFG, so wie in Bild 3.10 gezeigt, abfangen muss. Die Dauer der IFG ist mit $2 \cdot T_{Bit}$ minimal kurz gewählt. Im Gegensatz zum IRG, beeinflusst sie die Übertragungseffizienz nämlich stark (siehe Abschnitt 4.3.2). Dass diese Zeitspanne zum Puffern notwendig und ausreichend ist, wird in Kapitel 4 und 5 bewiesen – ein grober Anhaltswert für die „Frame“- und „Slot“-Fehler ist je eine Bitzeit.

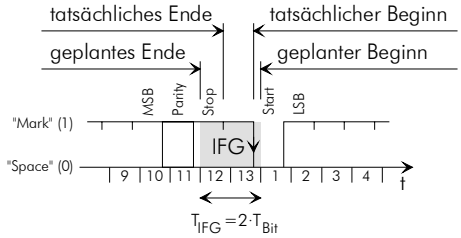


Bild 3.10: Pufferfunktion einer IFG

3.3 „Physical-Layer“

3.3.1 Klassifizierung

Für TTP/A gibt es verschiedene „Physical-Layer“, da das Protokoll universell und erweiterbar sein soll. In einfachen Anwendungen sind die Baudraten niedrig, die Störfestigkeit zweitrangig und der Kostendruck hoch. Anspruchsvolle Aufgaben verlangen dagegen hohe Baudraten und eine gute Störfestigkeit. Der Kostenaspekt ist zwar wichtig, er steht jedoch hinter der Funktion. Mit einem „Physical-Layer“ lässt sich dieses breite Spektrum nicht abdecken. Aus diesem Grund gibt es für TTP/A mehrere Vorschläge für den „Physical-Layer“. Wie Tabelle 3.4 zeigt, sind sie den drei allgemeinen Anwendungsklassen (niedrig, mittel, hoch) zugeordnet, da Geschwindigkeit, EMV und Kosten korrelieren. Eine ähnliche Klassifizierung hat die „Society of Automotive Engineers“ (SAE) aufgestellt (siehe Abschnitt 8.6). Die vorgeschlagenen „Physical-Layer“ basieren entweder auf vorhandenen Standards oder verwenden Standardbauteile. Näheres hierzu ist in den anschließenden Abschnitten nachzulesen. „Basic-TTP/A“ funktioniert mit allen aufgeführten „Physical-Layern“.

Anforderungs-klasse	Baudraten	Netzaus-dehnung	Übertragungs-standard	Kommentar
niedrig	≤ 20 kBit/s	bis 40m	ISO 9141	asymmetrische Übertragung; Eindrahtleitung ohne Schirm; Transistor- oder IC-Treiber; sehr günstig
mittel	≤ 1 MBit/s	bis 100m	RS485, CAN	symmetrische/differentielle Übertragung; Zweidrahtleitung einfach oder verdrillt; IC-Treiber; günstig
hoch	> 1 MBit/s	einige 100m	in der Entwicklung (siehe 3.3.3)	diff. Übertragung; synchrone Ternär-Codierung; verdrillte, geschirmte oder ungeschirmte Zweidraht-leitung oder LWL

Tabelle 3.4: „Physical-Layer“-Klassen in TTP/A

3.3.2 Standard-„Physical-Layer“

ISO 9141

„Low-Cost“-Systeme, wie z.B. Bedienterminals, Sitzheizungen oder Lichtsteuerungen, benötigen keine hohen Übertragungsgeschwindigkeiten und müssen weder sicher noch hoch zuverlässig funktionieren. In erster Linie müssen sie günstig sein und vernünftig arbeiten. Deshalb ist eine differentielle Übertragung, eine Zweidrahtleitung, eine Schirmung oder gar eine optische Verbindung zu teuer. Ein einfacher Draht, ein simpler Transistor und ein paar Widerstände müssen als „Physical-Layer“ ausreichen. Mit dieser Übertragungstechnik arbeitet der bekannte ISO 9141-Bus (siehe Punkt 2.3.2). Da er standardisiert ist und es preiswerte ICs für die Busanschaltung gibt, findet sein „Physical-Layer“ in TTP/A für einfache Systeme mit niedrigen Anforderungen Verwendung.

Aus praktischen Erfahrungen im Kfz-Bereich weiß man, dass mit dem ISO 9141-Bus Baudraten bis 20 kBit/s und Leitungslängen bis 40 m realisierbar sind. Die Signallaufzeiten können bei diesen Parametern vernachlässigt werden (siehe Abschnitt 4.1.6). Beim Einsatz des ISO 9141-„Physical-Layers“ sollte man die mäßigen EMV-Eigenschaften beachten. Zum einen ist die Übertragung asymmetrisch und zum anderen bildet der Signaldraht mit der Bezugsmasse häufig eine große Leiterschleife. Deshalb können Gleich- und Gegentaktstörungen (z.B. Masseversatz oder Stromimpulse auf Nachbarleitungen) Daten leicht verfälschen.

RS485/CAN

Systeme mit höheren Ansprüchen (Tempomat, Notaus, „Airbags“, Medizintechnik...) müssen unempfindlicher sein und i.A. zügiger reagieren. Der Kostenfaktor spielt eine Rolle, aber dominiert nicht. Das bedeutet, dass die Technik preiswert sein muss, aber nicht günstig. Anstelle der Eindrahtleitung mit asymmetrischer Übertragung, tritt eine Zweidrahtleitung mit symmetrischer Übertragung. Bei der Zweidrahtleitung kann es sich um eine verdrillte oder einfache Version handeln. Unabhängig davon ist sie in der Regel an den Enden mit ca. 120 Ω abgeschlossen. Gegenüber dem „Physical-Layer“ für Systeme mit niedrigen Anforderungen, ist die EMV dieses „Physical-Layers“ viel besser. Aufgrund der differentiellen Übertragung werden Gleichtaktstörungen unterdrückt. Es entsteht keine große Leiterschleife, insbesondere bei einer verdrillten Zweidrahtleitung. Hierdurch verringern sich induktive Einkopplungen erheblich. Zur Abschwächungen von kapazitiven Störungen trägt der symmetrische Aufbau bei. Bei starken Influenzstörungen können schaltungstechnische Maßnahmen[65] oder eine Schirmung Abhilfe schaffen.

Als etablierte Übertragungsstandards bieten sich RS485 und der CAN-„Physical-Layer“ an. Beide funktionieren ähnlich und können deshalb auch in einem heterogenen Netz betrieben werden [14]. Mit RS485 sind Geschwindigkeiten über 10 MBit/s bei max. 100 m Netzausdehnung realisierbar (Profibus). Der CAN-„Physical-Layer“ erreicht diese Werte nicht, weil er mit einer „Wired-AND“-Technik arbeitet. Allerdings vermag er mehr zu leisten als 1 MBit/s bei Netzausdehnung bis ca. 25 m. Diese Werte gelten nur für den CAN-Bus. Sie resultieren aus den Signallaufzeiten und scharfen Zeitrestriktionen der „In-Frame-Response“ (siehe Abschnitt 2.3.6). Ohne diese Bitarbitrierung kann man mit dem CAN-„Physical-Layer“ ebenfalls Distanzen von 100 m bei mehreren MBit/s überbrücken. Für TTP/A ist das mehr als genug. Schließlich handelt es sich um einen Sensor/Aktorbus. Wie in Kapitel 2 beschrieben, sind für diese Bussysteme Längen von 10..100 m und Baudraten von ein paar 10..100 kBit/s typisch. Im Forschungsprojekt wurde der „Physical-Layer“ RS485 im Demonetz (siehe Kapitel 5) und der CAN-„Physical-Layer“ beim Projektpartner SiemensVDO eingesetzt.

---

[65] Zum Beispiel Symmetrierkondensatoren und Mittelpunktwiderstände am Bus.

„Idle“-Zeiten

In den so genannten „Idle“-Zeiten, also während der IFG und IRG, belegt kein Knoten den Bus. Anhängig vom „Physical-Layer“ kann der Buspegel in diesem hochohmigen Zustand variieren. Die „Physical-Layer“ der Bussystem ISO 9141 und CAN arbeiten mit „Open Collector“-Technik. Sie kennen nur zwei Buszustände, nämlich nieder- und hochohmig bzw. dominant und rezessiv. Folglich ist der Buspegel in den „Idle“-Zeiten gleich dem hochohmigen Signalpegel. Damit „Basic-TTP/A“ funktioniert, muss dieser Signalpegel der logischen „1“ zugeordnet werden. Ansonsten finden die UARTs den Anfang der „Frames“ nicht. Bei einem UART-„Frame“ ist als Startbit eine logische „0“ und als Stoppbit eine logische „1“ definiert. Aufgrund dieses Unterschieds beginnt ein UART-„Frame“ immer mit einem „1-0“-Wechsel. Die Empfänger in den UARTs benötigen diese Startbitflanke für die „Frame“-Erkennung und Synchronisation. Aus diesem Grund müssen „Idle“-Zeiten für einen UART wie eine Verlängerung des Stoppbits aussehen. Bezüglich ihrer Spezifikation sind der CAN- und ISO 9141-Bus UART-konform („0“: niederohmig bzw. dominant; „1“: hochohmig bzw. rezessiv). Solange man also die entsprechenden Busanschaltungs-ICs verwendet, können keine Probleme auftreten. Bei „low-cost“ Knoten mit ISO 9141-„Physical-Layer“ wird die Busanschaltung aus Kostengründe jedoch oft diskret aufgebaut. Erfahrungsgemäß ist das eine Fehlerquelle, weil die notwendige Invertierung des Bitstroms gerne übersehen wird.

Neben den „Physical-Layern“ mit zwei Buszuständen, eignen sich für TTP/A auch die so genannten „Tristate-Physical-Layer“. Ein typischer Vertreter ist RS485. Bei ihm werden die logische „0“ und „1“ niederohmig gesendet. Die Umschaltung in den hochohmigen Zustand geschieht über ein separates Signal[66]. Aufgrund der Abschlusswiderstände sinkt der Buspegel im hochohmigen Zustand auf 0 V. Damit befindet er sich zwischen den „Schwellwerten“ für eine logische „0“ und „1“. Ohne zusätzliche Maßnahmen ist der Logikzustand bei RS485 in „Idle“-Zeiten also undefiniert. Aus den geschilderten Gründen ist das für TTP/A nicht zulässig. Deshalb muss RS485 soweit vorgespannt werden, dass der hochohmige Buspegel den Schwellwert für die logische „1“ überschreitet. Der Aufwand hierfür ist nicht groß, wie das folgende Beispiel mit zwei 470 Ω-Widerständen zeigt: Einer der Widerstände wird an die A-Leitung und V_{cc} (5 V) angeschlossen und der andere an die B-Leitung und GND. Dadurch entsteht eine Spannungsdifferenz $U_{AB} \approx 0{,}56$ V, die deutlich über der „1“-Schwelle von 0,2 V liegt [14]. In den praktischen Arbeiten traten mit dieser Erweiterung keine Probleme auf. Weitere Details enthalten die Schaltpläne unter Abschnitt 8.10.

Steckverbinder

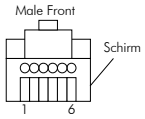
Auf Seiten der Anschlusstechnik sind in TTP/A bisher Standardsteckverbinder vom Typ RJ und Sub-D vorgesehen. Die preiswerten RJ-Steckverbinder eignen sich für „low-cost“ Anwendungen. Ihre Handhabung ist einfach und die Konfektionierung unproblematisch. Für TTP/A wurde die sechspolige Ausführung RJ11-6 ausgewählt. Die Pinbelegungen für die verschiedenen „Physical-Layer“ sind in Bild 3.11 dargestellt. Über die Pins 1, 3, 4 und 6 können die Knoten mit Energie versorgt werden. Allerdings muss man auf die Strombelastbarkeit der Kontakte Rücksicht nehmen. Ein Faustwert für den max. Effektivwert des Stroms pro Pin ist 1 A. Mit einem Einspeisepunkt können ca. 10..20 Knoten betrieben werden, sofern man Aktoren mit hohem Strombedarf separat versorgt. Typische und unproblematische Versorgungsspannungen liegen bei |Vcc| = 12 V (Kfz) oder |Vcc| = 24V (Industrie). Die Verwendung eines Kabelschirms ist optional[67].

---

[66] Als UART-Ereignis kann man entweder den „Transmit-“ oder „Receive-Interrupt“ (Mithören) verwenden. Ersterer tritt zu Beginn, Letzterer ungefähr in der Mitte des Stoppbits auf. Damit ist das Stoppbit mindestens zur Hälfte hochohmig.

[67] Erfahrungsgemäß ist die Wirkung eines Schirms bei digitalen Signalen besser, wenn man ihn beidseitig auflegt. Es dürfen jedoch keine Ausgleichsströme fließen. Außerdem muss ein Schirm großflächig kontaktiert werden.

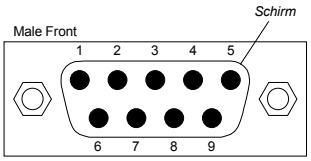
Anmerkung: Mit diesen Steckverbindern, ungeschirmten Kabeln und RS485 ist das Demonetz aufgebaut. Die sieben Knoten beziehen ihre Energie über Pin 1 (+12 V) und die GND-Pins. Auf Pin 6 befindet sich eine Hilfsspannung (-12V) für Knoten mit ADCs, DACs und OPVs.



PIN	RS485	CAN	ISO 9141
1	+Vcc	+Vcc	+Vcc
2	RS485-B	CAN-H	ISO-K
3	GND	GND	GND
4	GND	GND	GND
5	RS485-A	CAN-L	frei[68]
6	-Vcc	-Vcc	-Vcc

Bild 3.11: Belegung des RJ11-6-Steckers

Für Anwendungen in rauen Umgebungen (Produktionshalle, Motorraum) sind die robusteren Sub-D-Steckverbinder besser geeignet. In Anlehnung an den CAN-Bus [28], wurde für TTP/A die weitverbreitete neunpolige Sub-D-9-Version ausgewählt. Die Anschlussbelegung zeigt Bild 3.12.



PIN	RS485	CAN	ISO 9141
1	frei	frei	frei
2	RS485-A	CAN-L	frei[69]
3	GND	GND	GND
4	-Vcc	-Vcc	-Vcc
5	frei	frei	frei
6	GND	GND	GND
7	RS485-B	CAN-H	ISO-K
8	frei	frei	frei
9	+Vcc	+Vcc	+Vcc

Bild 3.12: Belegung des Sub-D-9-Steckers

3.3.3 „High-Speed-Physical-Layer“

Der Projektpartner SiemensVDO Automotive arbeitet an einem „Physical-Layer“ für hohe Anforderungen und Übertragungsgeschwindigkeiten > 1 MBit/s. Dieser so genannte „High-Speed-Physical-Layer“ ist für den Einsatz in Kfzs gedacht. Erste Versuche mit „Basic-TTP/A“ sind bereits erfolgreich durchgeführt worden; das Demonetz läuft mit dem ersten Entwurf des „High Speed Physical-Layers“. Dafür waren einige Anpassungen in der Implementierung notwendig, weil der „High-Speed-Physical-Layer“ die SPI-Schnittstelle[70] benötigt. Das „Basic-TTP/A“-Konzept ist davon aber nicht betroffen.

[68] Dieser Pin kann für eine redundante Datenleitung verwendet werden.

[69] dito

[70] SPI („Serial-Peripheral-Interface“) ist eine synchrone, serielle Schnittstelle. Bei den C16x-µCs heißt sie SSC („Synchronous-Serial-Interface“) und ermöglicht Baudraten bis zu 5 MBit/s.

Voraussichtlich wird der „High-Speed-Physical-Layer" Mitte 2002 fertiggestellt. Anwendung soll er in Kfz-Sicherheitseinrichtungen, wie „Crash"-Sensoren, „Airbag"-Zündkapseln, Gurtstraffern usw., finden.

Die anderen „Physical-Layer" von TTP/A sind dafür nur bedingt geeignet. Schon heute liegen die Realzeitanforderungen für z.B. Seiten-„Airbags" im Bereich von ca. 3 ms. Nach SiemensVDO müsste ein Bussystem heute bereits mit ca. 500 kBit/s arbeiten, um sämtliche Reaktionszeiten für alle Sicherheitseinrichtungen in einem Kfz garantieren zu können. Mit einer Verschärfung der Situation ist zu rechnen. In der Vergangenheit stiegen erstens die Anzahl der Rückhaltesysteme pro Kfz stetig an und zweitens die zeitlichen Anforderungen an sie. Zudem ist ein deutlicher Trend hin zu „X-by-Wire"-Systemen zu beobachten. Ein zukunftsorientierter Bus für diese Anwendungen hat laut SiemensVDO nur eine Chance, wenn er Baudraten deutlich über 1 MBit/s unterstützt. Außerdem müssen die Kosten so niedrig als möglich sein. Da ein genauer Oszillator teuer ist, sollen die „Slaves" über den Bus getaktet werden. Aus diesem Grund, und wegen den hohen Baudraten, hat SiemensVDO eine spezielle, synchrone Übertragung entwickelt. Sie arbeitet mit einer Ternärcodierung, zwei Standard-CAN-„Transceivern" und einer Codierlogik. Es laufen Versuche mit einer Kommunikation über das Kfz-Bordnetz (Kfz-„Powerline"). Grundlage hierfür ist der BST-„Physical-Layer" (siehe Abschnitt 2.3.5). Darüber hinaus wurde eine optische Übertragung (z.B. mit POFs[71]) und eine ASIC-Implementierung in Erwägung gezogen.

3.4 „Extended-Reliability"

Mit dem obligatorischen Baustein „Basic-TTP/A" kann man einfache Anwendungen realisieren. Für sicherheitskritische Anwendungen oder solche mit höheren Verfügbarkeitskriterien reicht das nicht aus. Dafür ist erstens die Restfehlerwahrscheinlichkeit von „Basic-TTP/A" zu hoch, besonders die der Daten. Zweitens existieren so genannte „Single-Points-of-Failure" bzw. „Golden-Units". Ohne einen „Master" arbeitet ein „Basic-TTP/A"-Netz nicht. Ein Kabeldefekt an der richtigen Stelle lässt das Netz ausfallen. Die Kommunikation ist nicht gegen „Bubbling-Idiots" geschützt. Und in manchen Anwendungen spielen bestimmte Sensoren und/oder Aktoren eine wichtige Rolle. Da „Basic-TTP/A" vor allem realzeitfähig ist, kann es für diese Fälle erweitert werden. Dazu muss die Redundanz erhöht werden. Sie richtig zu organisieren und zu verwalten ist die Aufgabe des optionalen TTP/A-Bausteins „Extended-Reliability", wodurch er prinzipiell eine Erweiterung für den „Data-Link-Layer" (2b) von „Basic-TTP/A" darstellt.

Der Baustein „Extended-Reliability" existiert zwar noch nicht, aber Kapitel 6 enthält einen konzeptionellen Vorschlag für ihn. Seine Umsetzung, Tests und gegebenenfalls sein Ausbau sollen Teil des geplanten Forschungsvorhabens „Smart-Transducers" werden (siehe Kapitel 7). Die Vorschläge sehen eine erweiterte Codierung für eine bessere Erkennung von Übertragungsfehlern vor, wobei auf die begrenzten Ressourcen von „low-cost" µCs Rücksicht genommen wird. Eine Fehlerkorrektur findet nicht statt, weil neben der Wahrscheinlichkeit für ein korrekt übertragenes Datum die Restfehlerwahrscheinlichkeit steigen würde. In den meisten Systemen ist es allgemein sinnvoller, über einen Fehler Bescheid zu wissen, um adäquat darauf reagieren zu können (z.B. „Fail-Safe", Extrapolation...). Die erweiterte Codierung erhöht die Hamming-Distanz auf mindestens sechs, wodurch die Restfehlerwahrscheinlichkeit, bei einer mittleren Bitfehlerwahrscheinlichkeit von 10^{-3}, weit unter 10^{-12} sinkt. Nach DIN 19244 (siehe Abschnitt 8.7) erreicht TTP/A damit die höchste Datenintegritätsklasse (I3: kritische Informationsübertragung).

[71] Eine „Plastic Optical Fiber" ist relativ günstig und eignet sich für kurze Entfernungen (10 m-Bereich). Man unterscheidet zwischen PCF-Technik (Glaskern) und APF-Technik (Kunststoffkern).

Neben dieser seriellen Redundanz sieht das Konzept eine parallele Redundanz vor. Auch hier liegt ein besonderes Augenmerk auf der Verwendung von Standardbauteilen und den Kosten. Deshalb müssen aufwändige, diversitäre Prinzipien ausgeschlossen werden. In einem ersten Schritt wird dem TTP/A-Bus ein zweiter „Master“ hinzugefügt, um die anfälligste „Golden-Unit“ zu beseitigen – in der Regel ist das der „Master“, weil er am komplexesten ist und aus den meisten Teilen besteht. Der TTP/A-Baustein „Extended-Reliability“ sorgt dafür, dass sich ein „Master“ aktiv und der andere passiv (Schatten-„Master“) verhält. Wenn der aktive „Master“ ausfällt, vertauscht der Baustein „Extended-Reliability“ die Rollen der beiden „Master“. Falls die Kosten es zulassen, kann man in einem weiteren Schritt ein zweites Netz hinzufügen[72]. Mit einem Parallelnetz besitzt TTP/A keinen „Single-Point-of-Failure“ mehr. Für die Verwaltung der beiden Busse und die notwendigen Interaktionen ist der TTP/A-Baustein „Extended-Reliability“ zuständig. Die weiterführenden Forschungsarbeiten werden zeigen, inwieweit diese konzeptionellen Vorschläge mit „low-cost“ µCs realisierbar sind.

3.5 „Interface-File-System”

Das ebenfalls optionale „Interface-File-System“ (IFS) ist der dritte TTP/A-Baustein. Wenn man es dem OSI-Referenzmodell gegenüberstellt, so ist es mit Schicht 7 („Application-Layer“) vergleichbar. Die Hauptaufgabe des IFSs ist der einheitliche und standardisierte Zugriff auf alle TTP/A-Daten. Dies umfasst die Prozessdaten, Informationen für die Diagnose oder Wartung, Konfigurations- und Dokumentationsdaten. Außerdem soll das IFS einen „Plug&Play“-Mechanismus beinhalten. Die notwendigen Voraussetzungen hält „Basic-TTP/A“ bereit. Es ist erweiterbar, dynamisch umkonfigurierbar und „hot-plugable“. Für den Transport von Nichtprozessdaten reserviert das IFS eine Runde (z.B. Runde 0). Denn im Gegensatz zu den Runden für die Prozessdaten, an denen i.A. viele Knoten teilnehmen, benötigt das IFS Punkt-zu-Punkt-Verbindungen zwischen den „Slaves“ und dem „Master“. Aufgrund dieser Eigenschaft, wird die reservierte Runde „Master/Slave“-Runde genannt. Die Arbeiten am IFS werden von den Projektpartnern Technischen Universität Wien und Universität Stuttgart durchgeführt. Sie sind noch nicht abgeschlossen und sollen ebenfalls in dem geplanten Forschungsvorhaben „Smart-Transducers“ zu Ende geführt werden.

Grundlage des IFSs ist eine statische Dateistruktur. Jede Datei kann bis zu 256 so genannte „Records“ enthalten. Alle „Records“ sind gleich aufgebaut und enthalten vier Daten- und ein horizontales „Check“-Byte. Der erste „Record“ einer Datei hat die Funktion eines „Headers“. Hingegen stellt der letzte „Record“ ein Konglomerat aus vertikalen „Check“-Bytes dar. Zusammen mit den horizontalen „Check-Bytes“, können Einbitfehler in einer Datei korrigiert werden. Der Zugriff auf einen bestimmten „Record“ erfolgt über ein Adressierungsschema. Jeder Knoten enthält eine 8 Bit-Adresse, jede Datei einen 6 Bit-Namen und jeder „Record“ einen 8 Bit-Index. Möchte man z.B. den „Record“ 3 in der Datei 7 im Knoten 20 lesen, so sendet der „Master“ in den ersten „Slots“ einer „Master/Slave“-Runde die Werte 20, 7 und 3, gefolgt von einem Lesekommando. Daraufhin schreibt der ausgewählte „Slave“ den Inhalt des gewünschten „Records“ in die nachfolgenden „Slots“ der „Master/Slave“-Runde. Das Schreiben in einen „Record“ oder das Ausführen einer RODL geschieht analog – Prozessdaten werden nach wie vor auf die „Basic-TTP/A“-Weise transportiert. In einem System können natürlich auch mehrere TTP/A-Netze existieren, die z.B. über einen Feldbus hierarchisch vernetzt sind. Wie in Abschnitt 3.2.1 beschrieben, fungieren die „Master“ dann zusätzlich als „Gateways“. Möchte man in einer solchen Baumstruktur auf einen bestimmten „Record“ zugreifen, so benötigt man ein erweitertes Adressierungsschema. Es muss ja das richtige „Sub“-Netz angesprochen werden. Deshalb

[72] In der Kfz-Technik lassen die Kosten kein Parallelnetz zu (z.B. BST-Bus). Sicherheitseinrichtungen müssen so konstruiert sein, dass sie günstig sind und ihr Versagen in der Lebenszeit eines Kfzs unwahrscheinlich ist („Fail-Operational“).

enthält das „Master“-IFS eine weitere Adresse, die so genannte „Cluster-ID“. Bild 3.13 enthält hierzu ein Beispiel des Projektpartners Universität Stuttgart [18]. Es handelt sich um die erwähnte Internet-Anbindung des Demonetzes über das IFS mit XML-Daten („Extended-Markup-Language“)[73].

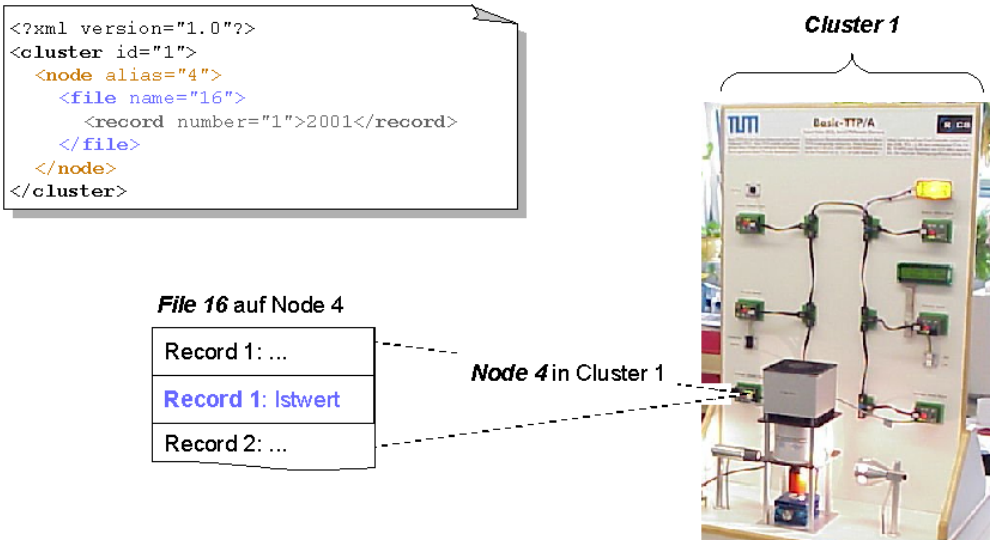


Bild 3.13: IFS/XML-basierter Datenzugriff auf das Demonetz

Rechts ist das Demonetz mit der „Cluster-ID“ 1 zu sehen und links das XML-Dokument für einen Schreibzugriff auf einen „Record“ in Knoten 4. Das XML-Dokument wird dem „Gateway-Master“ am Stück zugeschickt. An dem Eintrag <cluster id=“1“> erkennt er, dass es für ihn bestimmt ist. Demzufolge wertet der „Gateway-Master“ das XML-Dokument weiter aus. Die nächsten drei Einträge teilen ihm die „Slave“-Adresse 4 <node alias=“4“>, den Dateinamen 16 <file name=“16“> und den „Record“-Index 1 <record number=“1“> mit. An der Zahl 2001 erkennt der „Gateway-Master“ zum einen die Schreiboperation und zum anderen den Wert dafür. Nun prüft er, ob es sich um ein Prozessdatum oder eine anderweitige Information handelt. Im ersten Fall legt er den Wert in der RODL ab und stößt die entsprechende Runde an, damit das Datum seinen Zielort erreicht. Wenn das Datum jedoch der Diagnose, Wartung oder Konfiguration dient, initiiert der „Gateway-Master“ eine „Master/Slave“-Runde. Weitere Details können in [17] und [22] nachgelesen werden.

3.6 Zusammenfassung

In diesem Kapitel wurde das TTP/A-Protokoll vorgestellt. Es beschreibt eine für Bussysteme ungewöhnliche Architektur, die den neuen Sensor/Aktorbus skalierbar und offen macht. TTP/A besteht aus dem obligatorischen Baustein „Basic-TTP/A“ und den optionalen Bausteinen „Extended-Reliability“ und „Interface-File-System“. Das Basisprotokoll „Basic-TTP/A“ legt die grundlegenden Eigenschaften des Protokolls fest. Diese sind: kostengünstig, realzeitfähig, effizient und auf Standardbauteilen basierend. Der Baustein „Extended-Reliability“ ist für den Einsatz in verlässlichen Systemen erforderlich. Er sorgt für eine hohe Fehlersicherheit und unterstützt redundante Strukturen. „Plug&Play“, „Online“-Konfiguration, „Online“-Wartung, eine standardisierte Informationsdarstellung und der ein-

[73] Mit dem XML-Format kann man Daten und Informationen jeglicher Art einheitlich, strukturiert und leserlich darstellen. Des Weiteren ist das XML-Format zukunftsorientiert, weil eine direkte Internet-Anbindung möglich ist.

heitliche Zugriff auf Daten bietet das so genannte „Interface-File-System“. Der Grundstein für diese Architektur wurde in dieser Arbeit gelegt.

Des Weiteren leistet diese Arbeit wichtige Beiträge zum Basisprotokoll „Basic-TTP/A“. Bei der Konzipierung wurde darauf geachtet, dass „Basic-TTP/A“ mit „low-cost“ µCs mit und ohne Hardware-UARTs realisierbar ist. Ferner wurde auf die geringen Ressourcen von „low-cost“ µCs, wie Speicher, Rechenleistung, „Timer“ und Pins, Rücksicht genommen. Es war nötig eine implizite Synchronisation zu entwickeln und die Länge der IRG einem „Slot“ gleichzusetzen. Beides wirkte sich auch anderweitig positiv aus. Die implizite Synchronisation und eine drastische Verkürzung der IFG führten zur hohen Effizienz von „Basic-TTP/A“. Mit der gewählten und konstanten Länge der IRG konnte hingegen ein absoluter Zeitdeterminismus erzielt und die Basis für einfache Zeitberechnungen geschaffen werden.

Die Erarbeitung der „Fireworks“-Codierung stellt einen weiteren Beitrag zu „Basic-TTP/A“ dar. Sie ist auf die Rahmenbedingungen angepasst und bietet eine gute Fehlersicherung. Zusammen mit der Idempotenz der Daten qualifiziert sie das Basisprotokoll für die Datenintegritätsklasse I1 nach DIN 19244. Ferner ist dieser Arbeit zu verdanken, dass „Basic-TTP/A“ keine Unterschiede zwischen Prozess-, Konfigurations- und Diagnosedaten macht, alle 16 Runden gleichwertig sind und deshalb wenig „Overhead“ besitzt. Weitere Punkte sind die Einführung von sporadischen Runden, um den ereignisorientierten „Master/Slave“-Anteil hervorzuheben, die Definition unterschiedlicher Kommunikationsverbindungen durch entsprechendes „Scheduling“, Betrachtungen zur RODL-Implementierung und dem Ressourcenbedarf (insbesondere Speicher), das Aufzeigen der prinzipiellen inneren Abläufe und, zusammen mit SiemensVDO, die Festlegung verschiedene „Physical-Layer“.

4 „Basic-TTP/A“-Konzept

4.1 Probleme und Lösungsansätze

4.1.1 Zeitfehler

„Basic-TTP/A“ ist ein zeitgesteuertes Protokoll auf Basis einer asynchronen Kommunikation. Aus diesem Grund enthält das Protokoll keine direkten Zeitinformationen, weshalb jeder Busteilnehmer eine eigene Zeitbasis (Oszillator) besitzt. Es liegt in der Natur der Sache, dass diese unabhängigen Zeitbasen nicht exakt gleich sind, wodurch Fehler bei der „Frame“-Platzierung entstehen. Für die Kompensation dieser Fehler ist das IFG zuständig. Das bedeutet aber, dass die Zeitfehler gewisse Grenzen nicht übersteigen dürfen. Ansonsten kommt es zur Verletzung von Realzeitbedingungen, was in der Regel zu Buskollisionen und Datenverfälschungen führt. Deshalb werden in den anschließenden Ausführungen die möglichen Zeitfehler analysiert und Grenzen für sie spezifiziert.

Aus Abschnitt 3.2.4 ist bekannt, dass ein „Basic-TTP/A“-Knoten zwei Uhren benötigt. Eine für den UART-Bittakt und eine für die „Slot“-Vorhersage. Die UART-Uhr läuft eine relativ kurze Zeit frei ($11 \cdot T_{Bit}$), die „Slot“-Uhr dagegen eine relativ lange Zeit ($\leq 3289 \cdot T_{Bit}$). Aus diesem Grund fallen die Restriktionen für die „Slot“-Uhr deutlich schärfer aus als für die UART-Uhr. Der „Master“ ist hiervon weniger betroffen als die „Slaves“. Er verfügt stets über einen genauen Quarzoszillator. Die „Slaves“ müssen aus Kostengründen auch mit ungenauen RC- oder „On-Chip“-Oszillatoren arbeiten können. Außerdem ist der „Master“ die Zeitreferenz, wodurch er gegenüber sich selbst keinen Fehler aufweisen kann. Die einzige Forderung an ihn ist eine nahezu konstante Frequenz, um dieser Aufgabe gerecht zu werden. Schließlich gehen die „Slaves“ von einer gleichmäßig vergehenden Zeit aus. Als Richtlinie für den „Master“ gilt, dass seine Zeitvariationen (z.B. durch Temperaturschwankungen) bei einer max. langen Runde (255 „Slots“) wesentlich kleiner als $1/16 \cdot T_{Bit}$ sein müssen. Für einen Standardquarzoszillator stellt das kein Problem dar. Des Weiteren laufen die beiden Uhren im „Master“ fest gekoppelt, mit dem Teilungsverhältnis 13:1. In einem „Slave“ hingegen bewegt sich dieses Verhältnis um den Wert 13:1, weil dort die Uhren i.A. lose gekoppelt laufen.

Offensichtlich sind die „Slaves“ das eigentliche Problem bei der Einhaltung der Realzeitbedingungen. Und zwar nicht nur wegen der Frequenz- bzw. Gangfehler der Uhren, sondern auch wegen deren Phasen- bzw. Lagefehler. Aus diesem Grund müssen in einem „Slave“ vier mögliche Fehlerquellen (zwei Uhren à zwei Fehler) untersucht werden. Für die nachfolgenden Analysen werden quasistationäre Uhrenfehler angenommen. Denn die zeitlichen Änderungen der Oszillatorfrequenzen gehen, gegenüber typischer „Basic-TTP/A“-Zeiträume, sehr langsam vonstatten.

UART-„Timing“

Eine fehlerfreie UART-Kommunikation ist in einem gewissen Toleranzband für die Bitfrequenzen möglich. Als Anhaltspunkt hat sich ein Wert von etwa ± 5 % eingebürgert [133]. Diese Toleranzgrenzen sind jedoch nicht konstant, wie die folgenden Beispiele zeigen. Bei zwei Teilnehmern, minimal kurzen UART-„Frames“ (1 Startbit, 7 Datenbits, 1 Stoppbit) und einer Abtastung in Bitmitte, dürfen die Bittakte um $\pm 0{,}5 \cdot T_{Bit} / 8{,}5 \cdot T_{Bit} \cong \pm 5{,}9$ % auseinanderliegen. Sind die UART-„Frames“ dagegen maximal lang (1 Startbit, 8 Datenbits, 1 Paritybit, 2 Stoppbit), ergibt sich, unter sonst gleichen Bedingungen, ein Toleranzband von $\pm 0{,}5 \cdot T_{Bit} / 11{,}5 \cdot T_{Bit} \cong \pm 4{,}3$ %. Außer diesen bekannten UART-Parametern, spielen die Anzahl der Teilnehmer und Mehrfachbitabtastungen eine wichtige Rolle. Da die Toleranzgrenzen also stark schwanken können, ist deren exakte Berechnung, unter Berücksichtigung der „Basic-TTP/A“-Spezifika, unabdingbar.

Die Berechnung erfolgt anhand von Bild 4.1. Es sind die letzten Bits der „Frames“ von drei Knoten zu sehen. Auf den Zeitachsen befinden sich die Abtastpunkte. Oben sendet bzw. empfängt ein langsamer „Slave“, in der Mitte der „Master“ und unten ein schneller „Slave“. Des Weiteren werden eine Dreifachabtastung und achtfache Bitteilung angenommen. Heutzutage findet man, aus Gründen der Störfestigkeit, nur noch wenige UARTs mit einer Abtastung pro Bit[74]. Außerdem stellt eine Dreifachabtastung den „Worst-Case“ dar. Aus dem gleichen Grund wird mit einer achtfachen Bitteilung gerechnet. Üblich ist eine sechzehnfache Bitteilung, doch gerade im interessanten „low-cost“-Segment existiert der eine oder andere µCs mit achtfach abtastendem UART[75].

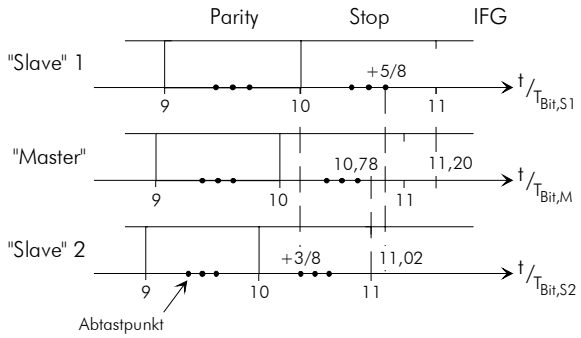


Bild 4.1: „Worst-Case“-Szenario für die UART-Frequenzabweichung in „Basic-TTP/A“

Nun zu den Details: Als Synchronisationspunkt verwendet ein UART die Startbitflanke. In diesem Moment interpretieren Sender und Empfänger die Zeit nahezu gleich. Danach beginnen ihre Uhren, entsprechend der Frequenzabweichung, auseinander zulaufen. Somit tritt der größte Zeitversatz am Ende eines „Frames“, also im Stoppbit, auf. Wenn nur zwei Knoten an einer Kommunikation teilnehmen, muss bei max. Frequenzabweichung und Idealbedingungen (unendlich steile Signalflanken, kein Abtast-„Jitter“, konstanter Bittakt in einem „Frame“) der letzte Abtastpunkt eines langsamen Empfängers („Slave“ 1) mit dem Ende des Stoppbits eines schnellen Senders („Slave“ 2) zusammenfallen. In „Basic-TTP/A“ ist die Sachlage ein klein wenig anders. Hier existiert die IFG, wodurch eine Startbitflanke nicht unmittelbar auf ein Stoppbit folgen kann. Unter Punkt 3.3.2 wurde gezeigt, dass die logischen Pegel des Stoppbits und des IFGs gleich sind. Deshalb darf der langsamere „Slave“ 1, ein „Frame“ des schnelleren „Slaves“ 2, auch noch in der IFG abtasten.

---

[74] In der Regel treffen UARTs bei einer Dreifachabtastung eine Mehrheitsentscheidung.

[75] Ein Beispiel ist der ST62x18 von STMicroelectronis. Er ist nicht besonders schnell, dafür aber gut ausgestattet.

In Bild 4.1 ist der letzte Abtastpunkt von „Slave“ 1 mit „+5/8“ gekennzeichnet. Wie man sieht, befindet sich dieser Punkt für „Slave“ 2 bei $11{,}02 \cdot T_{Bit}$, also sehr kurz nach dem Stoppbit. Diese Festlegung ergibt sich aus dem „Worst-Case“. Er entsteht bei „Basic-TTP/A“, wenn der erste Abtastpunkt des schnellen „Slaves“ 2 („+3/8“) auf den Beginn des Stoppbits vom langsamen „Slave“ 1 fällt – somit verhält sich „Basic-TTP/A“ dual zum Normalfall. Nachdem der „Worst-Case“ geklärt ist, lässt sich aus ihm folgende Bedingung ableiten:

$$10 \cdot T_{Bit,\,S1} \le \left(10 + \frac{3}{8}\right) \cdot T_{Bit,\,S2} \quad \Rightarrow \quad f_{Bit,\,max} = \frac{83}{80} \cdot f_{Bit,\,min}$$

Gemäß der Gleichung, ist eine relative Bittaktabweichung zwischen zwei „Slaves“ von $3/80 \cong 3{,}75\ \%$ zulässig[76]. Jeder weitere „Slave“ muss darüber hinaus $f_{Bit,\,min} \le f_{Bit,\,S} \le f_{Bit,\,max}$ erfüllen, um an der Kommunikation teilnehmen zu können. Die Relation legt das Toleranzband qualitativ fest, jedoch nicht quantitativ. Mit einer Bezugsgröße lässt sich dieses Problem beheben. Am besten geeignet ist der „Master“-Bittakt $f_{Bit,\,M}$, da er die Zeitreferenz ist. Wählt man für $f_{Bit,\,M}$ das arithmetische Mittel aus $f_{Bit,\,max}$ und $f_{Bit,\,min}$, so wie in Bild 4.1 dargestellt, dann stellen sich symmetrische Toleranzgrenzen für die „Slave“-Bittakte ein. Die wichtigsten Schritte bei der Berechnung und das Ergebnis sind nachstehend aufgeführt:

$$\text{I)} \quad \frac{f_{Bit,\,max}}{f_{Bit,\,min}} = \frac{83}{80}$$

$$\text{II)} \quad f_{Bit,\,min} \le f_{Bit,\,S} \le f_{Bit,\,max}$$

$$\text{III)} \quad f_{Bit,\,M} = \tfrac{1}{2}\left(f_{Bit,\,max} + f_{Bit,\,min}\right)$$

$$\frac{\text{II}}{\text{III}}\Bigg) \quad \frac{2}{1 + \dfrac{f_{Bit,\,max}}{f_{Bit,\,min}}} \le \frac{f_{Bit,\,S}}{f_{Bit,\,M}} \le \frac{2}{1 + \dfrac{f_{Bit,\,min}}{f_{Bit,\,max}}} \quad \rightarrow \quad \frac{160}{163} \le \frac{f_{Bit,\,S}}{f_{Bit,\,M}} \le \frac{166}{163}$$

$$\Rightarrow \quad \left|F_{UART}\right| = \left|\frac{f_{Bit,\,Slave} - f_{Bit,\,Master}}{f_{Bit,\,Master}}\right| \le 1{,}84 \cdot 10^{-2}$$

Formel 4.1: Maximale UART-Uhrenabweichung in „Basic-TTP/A“

Der relative Bittaktfehler eines „Slaves“, bezogen auf den „Master“-Bittakt, darf also höchstens ±1,84 % betragen[77]. In der Praxis sollte man dieses Toleranzband allerdings nicht ausreizen, da es ein paar Schmutzeffekte gibt. Stillschweigend wurden unendlich steile Signalflanken und eine perfekte Abtastung vorausgesetzt. Tatsächlich ist die Steilheit der Signalflanken begrenzt, was insbesondere bei höheren Baudraten zu einer Verkürzung der eindeutigen Bitpegelzeiten führt. Außerdem können die Abtastpunkte ein wenig „jittern“. Die Steuerung eines UART ist i.A. ein Zustandsautomat, der ent-

[76] Bei zwei Teilnehmern und ohne IFG wären es ungefähr ±3,5 %. Dieses Intervall ist etwa doppelt so lang.

[77] Ohne die Verlängerung des Stoppbits durch das IFG ergäbe sich ein Wert von 1,76 %.

sprechend seiner Bitteilung mit $T_{Bit}/16$ oder $T_{Bit}/8$ getaktet wird. Wenn eine Startbitflanke zwischen zwei Taktzeitpunkten erscheint, registriert sie der UART erst beim nächsten Takt. Dadurch kann ein Abtastfehler von $0..T_{Bit}/n$ (n: Bitteilung) entstehen. Aus diesen Gründen ist in praktischen Anwendungen ein Toleranzband von ca. ±1,6..1,7 % empfehlenswert. Für die Untersuchung der Zeitfehler zählt jedoch das theoretische Toleranzband von ±1,84 %. Hieraus kann die max. Schwankungsbreite der „Frames“ ermittelt werden. Ein „Slave“ an der unteren Toleranzgrenze sendet „Frames“ der Dauer $11 \cdot T_{Bit, M} \cdot 0{,}9816 \approx 10{,}78 \cdot T_{Bit, M}$. Dagegen benötigt ein „Slave“ an der oberen Toleranzgrenze $11 \cdot T_{Bit, M} \cdot 1{,}0184 \approx 11{,}20 \cdot T_{Bit, M}$. Für die max. Überlänge von $0{,}20 \cdot T_{Bit, M}$ muss aber kein Platz in der IFG reserviert werden. Es wird sich nämlich zeigen, dass man die „Frames“ abschneiden kann.

Als erstes wird der Empfang von „Frames“ behandelt. Der letzte Abtastpunkt entsteht durch einen minimal langsamen „Slave“. Er ist in Bild 4.1 mit „+5/8“ gekennzeichnet. Solange der Schnitt nicht vor diesem Punkt liegt, empfangen alle „Slaves“ einen gestutzten „Frame“ korrekt. Des Weiteren können alle „Slaves“ den nächsten „Frame“ kurz nach dem Zeitpunkt „+5/8“ empfangen, da für einen UART der Empfangszyklus mit dem letzten Abtastpunkt zu Ende und seine Startbiterkennungen wieder aktiv ist. Beim Senden muss man zwei Fälle unterscheiden. Im Ersten soll der minimale langsame „Slave“ ein „Frame“ senden, das von einem „Frame“ eines anderen Knotens, ab dem Schnittzeitpunkt „+5/8“, überschrieben wird. Elektrisch gesehen entstehen keine Probleme, weil alle Knoten das Stoppbit spätestens ab dem letzten Abtastpunkt hochohmig übertragen. Dieser Grenzfall trifft für „Tristate-Physical-Layer“ (RS485) und UARTs ohne „Transmit Interrupt“ zu. In der Regel erfolgt die Umschaltung in den hochohmigen Zustand am Anfang des Stoppbits (siehe Abschnitt 3.3). Bei „Physical-Layern“ mit zwei Buszuständen (ISO 9141, CAN) ist das Stoppbit von Haus aus hochohmig.

Der zweite Fall ist ähnlich gelagert. Wieder soll der „Frame“ des langsamen „Slaves“ ab dem Zeitpunkt „+5/8“ überschrieben werden. Im Unterschied zu vorher, stammen aber beide „Frames“ von ihm. Demzufolge müsste der langsame „Slave“ seinen Sendezyklus abbrechen. Allerdings ist das bei der überwiegenden Mehrzahl der UARTs nicht möglich. Ein Standard-UART kann den zweiten „Frame“ erst nach dem Ende des ersten „Frames“ senden, sofern das „Transmit“-Register ein Datum enthält. Deshalb muss man davon ausgehen, dass der langsame „Slave“ frühestens ab dem Zeitpunkt $11{,}20 \cdot T_{Bit, M}$ wieder senden kann. Glücklicherweise stört dieser Sendeverzug nicht. Denn die i.A. nicht vollständige Steuerbarkeit des UART-Sendeteils bewirkt einen „Frame-Jitter“ von bis zu einer Bitzeit, wodurch der Sendeverzug verdeckt wird – es kommt zu keiner Addition. Folglich können die „Frames“ ab dem Zeitpunkt „+5/8“ in Bild 4.1 problemlos abgeschnitten werden. Diese Erkenntnis ist für die Minimaldimensionierung der IFG wichtig.

Neben dem Längenfehler kann ein UART auch einen Lagefehler produzieren. Es handelt sich um den bereits erwähnten „Frame-“ bzw. UART-„Jitter“. Auf seine Ursache wird in Abschnitt 4.2.4 detailliert eingegangen. Für die Analyse der Zeitfehler zählt nur das Phänomen. Nachdem man das „Transmit“-Register eines UARTs beschrieben hat, beginnt derselbe mit dem Senden der „Frame“-Bits. Er macht das allerdings nicht immer sofort, sondern um bis zu eine Bitzeit verzögert (siehe Bild 4.2). Bei einer konventionellen UART-Kommunikation (z.B. PC-„Link“) stört das in der Regel nicht. Dort gibt es praktisch keine Realzeitanforderungen. Außerdem können mit den oft verwendeten Vollduplex-Verbindungen (z.B. RS232) keine Buskollisionen entstehen. In „Basic-TTP/A“ dagegen sind die Realzeitanforderungen hoch und es besteht die Gefahr einer Buskollision, wenn ein „Frame“ außerhalb seines „Slots“ liegt. Die vorangehenden Ausführungen haben zwar gezeigt, dass eine kleine Überschneidung der „Frames“ möglich ist. Für den UART-„Jitter“ reicht das jedoch nicht aus. Im „Worst-Case“ sendet ein minimal langsamer „Slave“ einen „Frame“ mit maximaler Verzögerung. Er benötigt dafür zwölf Bitzeiten nach seiner UART-Uhr. Durch das Stutzen der „Frames“, kann diese Zeitspanne auf $(11 + 5/8) \cdot T_{Bit, S} = 11{,}625 \cdot T_{Bit, S}$ reduziert werden. In Referenzzeit umgerechnet („Master“) ergibt

das $11{,}625/(0{,}9816 \cdot f_{Bit,\,M}) \approx 11{,}84 \cdot T_{Bit,\,M}$. Somit benötigt man für die Kompensation der beiden UART-Fehler $0{,}84 \cdot T_{Bit,\,M}$, von der zwei „Master"-Bitzeiten langen IFG.

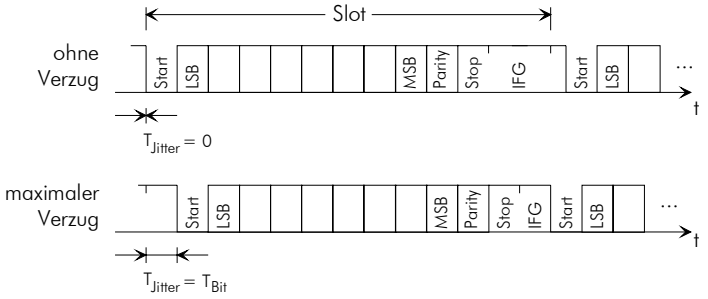


Bild 4.2: UART-„Jitter" in „Basic-TTP/A"

„Slot-Timing"

Für die zeitliche Platzierung der UART-„Frames" sind die „Slot-Timer" verantwortlich. Da der „Master" die Zeitreferenz ist, läuft sein „Slot-Timer" frei. Die „Slaves" hingegen haben ein Synchronisationsproblem. Wegen der asynchronen Kommunikation, erhalten sie keine direkten Zeitinformationen vom „Master". Deshalb müssen die „Slaves" ihre „Slots" vorhersagen. Als einzige Orientierungsmarken dienen ihnen die „Fireworks-Frames". Sie treten in einer gewissen Regelmäßigkeit auf, weil die Runden nicht beliebig lang sind. Außerdem sorgt der „Master" mit speziellen Maßnahmen (siehe Abschnitt 4.2) für einen geringen „Jitter" der „Fireworks-Frames". Dadurch wird den „Slaves" eine implizite, effizienzneutrale Synchronisation ermöglicht. Die grobe Funktionsweise und die Probleme sollen anhand von Bild 4.3 erörtert werden.

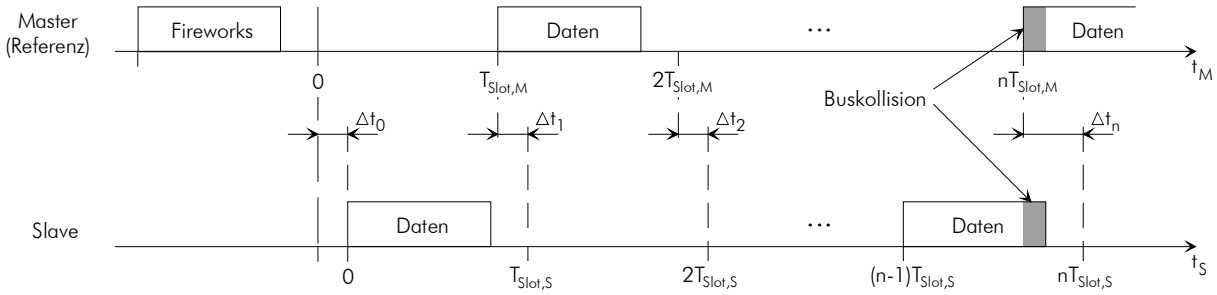
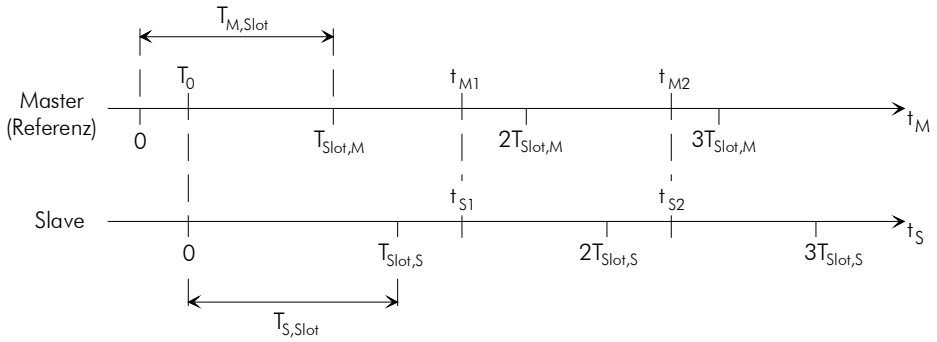


Bild 4.3: Funktionsweise und Probleme bei der „Slot"-Vorhersage (ohne UART-Fehler)

Vom Prinzip funktioniert die „Slot"-Vorhersage so ähnlich wie die Bitvorhersage in einem UART. Die „Slot-Timer" in den „Slaves" weisen, gegenüber dem „Master", einen Gangfehler auf. Mit präzisen Oszillatoren oder einem Korrekturalgorithmus lässt sich der Gangfehler zwar einschränken, aber nicht beseitigen. Folglich kommt es zu einer Gangfehlerakkumulation, wodurch der „Slot-Timer"-Fehler stetig wächst. Des Weiteren haben die „Slaves" ein Problem nach dem „Booten". Hier können die „Slot-Timer" einen großen Lagefehler aufweisen, weil den „Slaves" jegliche Orientierung fehlt. Aus diesen Gründen müssen die „Slot-Timer" in regelmäßigen Abständen neu gestellt werden. Diese so genannten Nullpunktabgleiche führen die „Slaves" an den „Fireworks-Frames" durch. Gemäß Bild 4.3, ist als Nullpunkt die Grenze zwischen dem „Fireworks-" und erstem Daten-„Slot" definiert, da ab diesem Zeitpunkt die „Slot"-Vorhersage in einer Runde beginnt.

Leider kann der Nullpunktabgleich nicht perfekt durchgeführt werden. Deshalb existiert neben dem Gangfehler Δf auch ein Lagefehler Δt_0. Er ist mit dem Fehler bei der Startbiterkennung in einem UART vergleichbar. Allerdings darf der Lagefehler Δt_0 in den meisten Fällen nicht vernachlässigt werden. Denn er besitzt eine unangenehme Eigenschaft. Sein relativer Anteil am Gesamtfehler nimmt mit steigender Baudrate zu. Dafür sind in erster Linie „Interrupt“-Latenzzeiten, Programmausführungszeiten und Signallaufzeiten verantwortlich. Sie bilden im Lagefehler Δt_0 einen konstanten Term, wohingegen die IFG umso kleiner wird, je höher die Baudrate ist. Ohne Gegenmaßnahmen verkleinert sich der Spielraum bereits ab Baudraten von 50..100 kBit/s soweit, dass er selbst für „Slaves“ mit Quarzoszillatoren oft nicht mehr ausreicht (siehe Bild 4.3). Deshalb benötigt ein „Slave“ einen zusätzlichen Korrekturmechanismus zur Verringerung des „Slot-Timer“-Lagefehlers (siehe Abschnitt 4.1.2). Adäquates für den Gangfehler (siehe Abschnitt 4.1.3) ist dagegen nur bei „Slaves“ mit ungenauen Oszillatoren (RC oder „On-Chip“) erforderlich.



$$t_M(t_S) = m \cdot t_S + k \qquad \text{mit} \qquad m = \frac{t_{M2} - t_{M1}}{t_{S2} - t_{S1}} = \frac{T_{Slot,S}}{T_{Slot,M}}$$

$$\text{und} \qquad k = \frac{t_{M1} \cdot t_{S2} - t_{S1} \cdot t_{M2}}{t_{S2} - t_{S1}} = T_0$$

Formel 4.2: Fehlermodell für die Transformation einer „Slave“-Zeit in „Master“-Zeit

Zur Beurteilung der „Slot-Timer“-Restfehler müssen ein Fehlermodell aufgestellt und Toleranzgrenzen festgelegt werden. Zuerst das Fehlermodell: Der Lagefehler Δt_0 ist während einer Runde konstant. Zu ihm addiert sich pro Zeiteinheit der Gangfehleranteil $dt_{Gang} = (\Delta f(t)/f) \cdot dt$. Folglich gilt für den Gesamtfehler des „Slot-Timers“ der allgemeine Ansatz: $\Delta t_{ges} = \Delta t_0 + 1/f \cdot \int \Delta f(t) \cdot dt$. Da die Frequenzdrift $d(\Delta f(t))/dt$ innerhalb einer Runde sehr gering ist, darf man von einem quasistationären Gangfehler $\Delta f \approx$ konst. ausgehen. Die Frequenzdrift entsteht durch den „Master“ und den jeweiligen „Slave“. Verantwortlich sind vor allem Temperaturschwankungen, deren Zeitkonstanten typischerweise im oberen s-Bereich liegen. Eine Runde hingegen dauert nur wenige 10 ms, weshalb sich die Temperatur in dieser Zeit kaum ändern kann. Des Weiteren spielt der Temperaturdurchgriff $df/d\upsilon$ eine Rolle. Er ist beim „Master“ mit Quarzoszillator sehr klein und bei einem „Slave“ mit RC- oder „On-Chip“-Oszillator klein[78]. Deshalb ist folgender, approximierender, linearer Ansatz zwischen einer „Slave“-Zeit t_S und der „Master“-Zeit t_M zulässig: $t_M(t_S) = m \cdot t_S + k$. Er bildet die Basis für eine Interpretation des relativen „Slot-Timer“-Fehlers: $F_{Timer} = (t^*_S - t_M)/t_M$ mit $t^*_S = t_M(t_S) = m \cdot t_S + k$. Die Bestimmung der Parameter k

[78] Für Quarzoszillatoren o.Ä. ist $df/d\upsilon \approx 10^{-8}$.. 10^{-7} 1/K und für RC-Oszillatoren o.Ä. $\approx 10^{-5}$.. 10^{-4} 1/K.

und m enthält Formel 4.2. Je nach Anwendungsfall, kann man entweder mit zwei Punktepaaren (t_{M1}, t_{S1}) und (t_{M2}, t_{S2}) oder den Größen T_0, $T_{Slot,\,M}$ und $T_{Slot,\,S}$ arbeiten – Letztere sind i.A. leichter messbar.

Nun zu den Toleranzgrenzen: Für eine Quantifizierung benötigt man Angaben über die Vorhersagezeit und den Spielraum in der IFG. Eine Runde kann max. 255 „Slots" lang werden. Da die „Slot"-Vorhersage mit dem ersten Daten-„Slot" beginnt und am Anfang des letzten Daten-„Slots" endet, muss ein „Slave" höchstens eine Zeitspanne von $253 \cdot T_{Slot,\,S}$ richtig abschätzen. Als Spielraum verbleiben, abzüglich von $0{,}84 \cdot T_{Bit,\,M}$ für die UART-Fehler, $1{,}16 \cdot T_{Bit,\,M}$ von der IFG. Davon werden $0{,}16 \cdot T_{Bit,\,M}$ für die Signallaufzeiten reserviert (siehe Abschnitt 4.1.6), weil ihre Korrektur für viele „low-cost" µCs zu aufwändig ist – insbesondere der Protokollaufwand. Somit verbleibt für die Kompensation der restlichen „Slot-Timer"-Fehler eine „Master"-Bitzeit ($T_{Bit,\,M}$). Sie muss um den Nominalwert $n \cdot T_{Slot,\,M}$ verteilt werden, da es schnellere und langsamere „Slaves" geben kann. Die Grenzen sind aber nicht exakt symmetrisch. Wie der nächste Unterpunkt zeigen wird, liegen sie bei etwa $(-6..10)/16 \cdot T_{Bit,\,M}$. Für eine allgemeine Interpretation sind jedoch symmetrische Fehlergrenzen besser. Aus diesem Grund, wegen des geringen Unterschieds und der gleichen Fehlerspannweite wird ein symmetrisches Toleranzband spezifiziert. Mit Hilfe von Formel 4.2 entsteht daraus:

$$F_{Timer,\,max} = \frac{\Delta t^{*}_{S,\,max}}{t_{M,\,max}} = \frac{\pm 0{,}5 \cdot T_{Bit,\,M}}{253 \cdot 13 \cdot T_{Bit,\,M}} \approx \pm 152 \cdot 10^{-6}$$

$$F_{Timer,\,max} = \frac{t^{*}_{S,\,max} - t_{M,\,max}}{t_{M,\,max}} = \frac{m \cdot t_{S,\,max} + k - t_{M,\,max}}{t_{M,\,max}}$$

$$\Rightarrow \qquad \left| \frac{T_{Slot,\,M}}{T_{Slot,\,S}} \cdot \left(1 - \frac{T_0}{t_M} \right) - 1 \right| \leq 152 \cdot 10^{-6}$$

Formel 4.3: Symmetrierter, relativer, zulässiger Betragsfehler des „Slot-Timers"

Das Ergebnis zeigt sehr schön die Wirkung der beiden „Slot-Timer"-Fehler. Bei kurzen Runden und hohen Baudraten (t_M klein) spielt der Lagefehler T_0 die Hauptrolle. Sind die Runden dagegen lang und die Baudraten niedrig (t_M groß), so dominiert der Gangfehler $T_{Slot,\,M}/T_{Slot,\,S}$. Außerdem spiegelt der Zahlenwert die hohen Ansprüche an das „Slot-Timing" wider. Im Vergleich zum UART-„Timing", muss es um den Faktor $1{,}84 \cdot 10^{-2} / 152 \cdot 10^{-6} \approx 121$ genauer sein. Dieser Wert relativiert sich bei einer Gegenüberstellung mit einem Standardquarzoszillator. Mit einer typischen Abweichung von $\pm 50 \cdot 10^{-6}$ (siehe Abschnitt 8.4) ist er ca. dreimal genauer als das „Slot-Timing". Daraus sieht man, dass eine Umsetzung zwar nicht trivial, aber möglich ist.

4.1.2 „Slot-Timer"-Lagefehlerkorrektur

Ursachen

Für die Entwicklung einer Korrekturmethode müssen die Zusammenhänge bekannt sein, die zu dem beschriebenen Lagefehler führen. Dazu ist seine prinzipielle Entstehung, bei Verwendung von Hardware-UARTs, in Bild 4.4 dargestellt[79]. Auf der obersten Zeitachse ist der „Fireworks-Frame" zu se-

[79] Bei einer UART-Emulation können sich ein paar Details ändern, jedoch nicht das Prinzip.

hen. Damit er nicht „jittert“, muss ihn der „Master“ etwas verzögert senden. Die genauen Gründe werden unter Punkt 4.2.4 erläutert. Hier wird der „Fireworks“-Verzug $T_{D,\,FW}$ („Delay“, „Fireworks“) als Tatsache behandelt. Seine Länge hängt von der „Master“-Implementierung ab. Normalerweise beträgt $T_{D,\,FW}$ etwa $\frac{1}{2}\cdot T_{Bit,\,M}$; ein Beispiel liefert die C16x-Implementierung mit $7/16\cdot T_{Bit,\,M}$. Ohne weiteres Zutun merkt der „Slave“ davon nichts, weshalb $T_{D,\,FW}$ der erste Beitrag zum Lagefehler T_0 ist. Bei UART-Emulationen ist $T_{D,\,FW}$ i.A. kleiner, dafür besteht die Möglichkeit eines „Jitter“ $\Delta T_{D,\,FW}$ von ungefähr $0..1/16\cdot T_{Bit,\,M}$.

Einen weiteren Beitrag bildet die Signallaufzeit $T_{D,\,Sig}$. Sie setzt sich aus der Leitungslaufzeit und Verzögerungen in „Transceivern“, Opto-Kopplern o.Ä. zusammen. Die Wirkung der Signallaufzeit auf den Lagefehler des „Slot-Timers“ ist dieselbe wie die des „Fireworks“-Verzug. Allerdings ist $T_{D,\,Sig}$, bei den typischen, geringen Sensor/Aktorbuslängen, wesentlich kleiner als $T_{D,\,FW}$; in Bild 4.4 ist $T_{D,\,Sig}$ größer gezeichnet, damit man es sieht. Ein 20 m langes Kupferkabel[80] z.B. verursacht eine Laufzeit von etwa 100 ns. Das entspricht bei einer Baudrate von 625 kBit/s, in Bitzeiten ausgedrückt, $1/16\cdot T_{Bit}$.

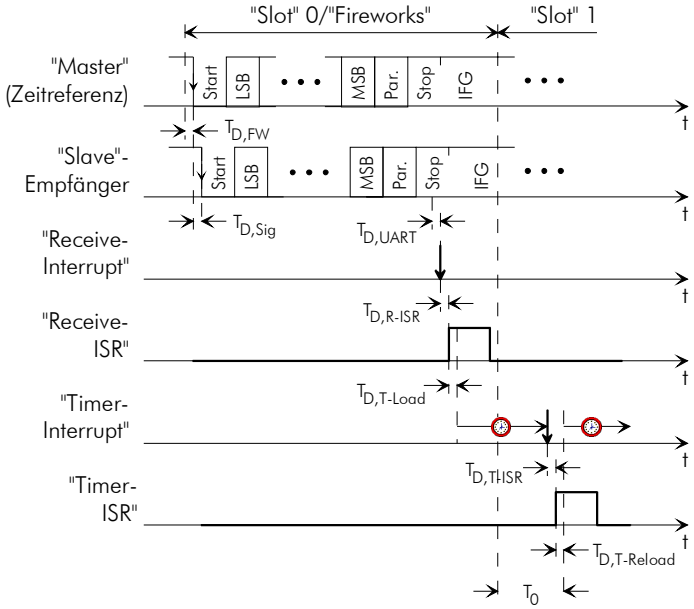


Bild 4.4: Prinzipielle Entstehung des „Slot-Timer“-Lagefehlers mit Hardware-UARTs

In der dritten Zeitachse von oben ist der nächste Fehler $T_{D,\,UART}$ dargestellt. Er ist eine Folge der unterschiedlichen Baudraten, wodurch sich der „Receive-Interrupt“ in einem „Slaves“ ein Stück verschiebt. Anhand eines Beispiels soll der Variationsbereich abgeschätzt werden. Bei den meisten UARTs tritt der „Receive-Interrupt“ nach dem letzten Abtastwert im Stoppbit auf[81]. Des Weiteren wird von einem UART mit achtfacher Bitteilung und drei Abtastungen in Bitmitte ausgegangen („Worst-Case“). Somit liegt der Sollzeitpunkt $(10 + 5/8)\cdot T_{Bit,\,M} \approx 10{,}625\cdot T_{Bit,\,M}$ nach der Startbitflanke. Gemäß Abschnitt 4.1.1, beträgt die zulässige Baudraten-Toleranz ±1,84 %. Demnach entsteht ein Variationsbereich $T_{D,\,UART}$ von ungefähr $\pm 0{,}2\cdot T_{Bit,\,M}$. Er kann durch den möglichen Fehler bei der Startbitflankenerkennung ($\Delta T_{D,\,UART} = 0..1/8\cdot T_{Bit,\,M}$) um $+0{,}125\cdot T_{Bit,\,M}$ größer ausfallen.

[80] Als Richtwert für die Ausbreitungsgeschwindigkeit bei einem Kupferkabel gilt: $v \approx 200\cdot 10^3$ km/s.

[81] Es gibt nur wenige UARTs, wo der „Receive-Interrupt“ am Ende des Stoppbits auftritt (z.B. im µC ST62x18).

Nach dem „Receive-Interrupt“ läuft die zugehörige „Interrupt-Service-Routine“ (ISR) an. Das geschieht nicht unmittelbar, sondern mit einer Verzögerung $T_{D,\,R\text{-}ISR}$. In dieser so genannten „Receive-Interrupt“-Latenzzeit rettet die CPU den „Task“-Kontext und lädt die neue Einsprungadresse. Da die Kontextgröße und die Befehlsausführungszeiten implementierungs-, „Compiler“- und CPU-abhängig sind, fällt $T_{D,\,R\text{-}ISR}$ unterschiedlich aus. Einen Anhaltswert liefert der C16x-µC. Je nach „Interrupt“-Modell, Speichermodell und „Memory Wait States“, kann $T_{D,\,R\text{-}ISR}$ zwischen 0,3..1 µs betragen. Eine Variation zur Laufzeit können dagegen „Critical-Sections“ bewirken[82]. Das Gleiche gilt prinzipiell für „Caches“, „Buffers“ und „Pipelines“. Von diesen Beschleunigungseinheiten findet man in „low-cost“ µCs aber nur „Pipelines“ (z.B. PIC- und AVR-µCs). Sofern eine CPU unterschiedliche Befehlsausführungszeiten besitzt (CISC), können sie die Variation der „Interrupt“-Latenzzeit erhöhen – Befehle wie „Reset“, „Sleep“, „Halt“ o.Ä. sind ausgeschlossen. Gegenüber einer „Critical-Section“, ist der Beitrag einer „Pipeline“ aber eher gering. Denn in den meisten „low-cost“ µCs arbeiten RISC- bzw. RISC-ähnliche CPUs, wodurch die Befehlsausführungszeiten konstant bzw. nahezu konstant sind. Außerdem haben die „Pipelines“ in „low-cost“ µCs lediglich zwei Stufen („Fetch“ und „Execute“), so dass höchstens ein längerer Befehl zu Ende geführt werden muss. Dadurch ergibt sich in der Regel eine Verzögerung von einem Maschinenzyklus; bei einer „Critical-Section“ können es auch zwei oder drei sein. Als Anhaltspunkt für die Variation $\Delta T_{D,\,R\text{-}ISR}$ bei höheren Baudraten, denn nur dort spielen sie eine Rolle, dient der C16x-Wert von $\approx 0..1/16 \cdot T_{Bit,\,M}$.

In der „Receive-ISR“ wird, neben der Prüfung des „Fireworks-Frames“ und der Rundenvorbereitung, der Nullpunktabgleich durchgeführt. Dafür wird der „Slot-Timer“ so aufgezogen, dass er nach Ablauf den Beginn des ersten Daten-„Slots“ möglichst genau trifft. Wie in Abschnitt 4.2.5 noch gezeigt wird, können die entsprechenden Befehle nicht am Anfang der ISR stehen. Deshalb muss die Verspätung $T_{D,\,T\text{-}Load}$ eingeplant werden. Ihre Größenordung liegt bei einer Interrupt“-Latenzzeit. Leider ist eine genauere Quantifizierung nicht möglich, weil $T_{D,\,T\text{-}Load}$ stark von der CPU, dem „Compiler“ und der Implementierung abhängt. Mit dem Programmkonzept aus Abschnitt 4.2, weist $T_{D,\,T\text{-}Load}$ aber zumindest keine Varianz auf.

Nach Ablauf des „Slot-Timers“ aktiviert dessen „Interrupt“ die „Timer-ISR“. Auch hier passiert das nicht sofort, sondern um die „Timer-Interrupt“-Latenzzeit $T_{D,\,T\text{-}ISR}$ verzögert. Da sich die „Interrupt“-Latenzzeiten normalerweise nicht bzw. nur kaum unterscheiden[83], gilt für $T_{D,\,T\text{-}ISR}$ und $\Delta T_{D,\,T\text{-}ISR}$ das Gleiche wie für $T_{D,\,R\text{-}ISR}$ und $\Delta T_{D,\,R\text{-}ISR}$. In der „Timer-ISR“ wird der „Slot-Timer“ auf den nächsten Daten-„Slot“ eingestellt, womit die eigentliche „Slot“-Vorhersage für einen „Slave“ beginnt – andere Aufgaben sind momentan nebensächlich. Für das Nachladen muss wiederum mit einer Verzögerung gerechnet werden. In Bild 4.4 ist sie mit $T_{D,\,T\text{-}Reload}$ berücksichtigt. Die Dauer liegt ebenfalls in der Größenordung der „Interrupt“-Latenzzeit. Außerdem muss auch hier mit keiner Varianz gerechnet werden.

Der letzte Fehler entsteht durch die Quantisierung ΔT_S des „Slot-Timers“. Aus Gründen der Übersichtlichkeit ist er nicht mehr in Bild 4.4 eingezeichnet. Unter der sinnvollen Annahme von gerundeten „Slot-Timer“-Werten (siehe Formel 4.7), beläuft sich der Quantisierungsfehler auf $\pm\frac{1}{2} \cdot \Delta T_S$. Gemäß Abschnitt 4.1.3 gilt für die „Slot-Timer“-Auflösung: $\Delta T_S \leq T_{Bit,\,M}/8$. Somit beträgt der Maximalwert des Quantisierungsfehlers $\pm T_{Bit,\,M}/16$. Da sich die aufgezählten Fehler summieren, entsteht bei einem Nullpunktabgleich ohne Korrektur ein „Slot-Timer“-Lagefehler von:

---

[82] „Critical-Sections“ sind nicht unterbrechbare Codestücke. Sie bilden eine Möglichkeit der Synchronisation beim Zugriff mehrerer „Tasks“ auf eine gemeinsame Ressource. Typische Beispiele sind „Read-Modify-Write“-Operationen oder die Adressierung von „Far“-Speicher. Letzteres dauert beim C16x-µC zwei bis drei Befehle und kommt des Öfteren vor.

[83] Bei den externen „Interrupts“ unterscheidet z.B. der C16x-µC zwischen Standard- und „Fast-Interrupts“. Die Geschwindigkeitsunterschiede sind jedoch nicht gravierend.

$$T_0 = T_{D,\,FW} + T_{D,\,Sig} + T_{D,\,UART} + T_{D,\,R-ISR} + T_{D,\,T-Load} + T_{D,\,T-ISR} + T_{D,\,T-Reload} + \Delta T_{D,\,FW} + \Delta T_{D,\,UART} + \Delta T_{D,\,R-ISR} + \Delta T_{D,\,T-ISR} \pm \tfrac{1}{2}\Delta T_S$$

Formel 4.4: „Slot-Timer“-Lagefehler ohne Korrektur

Unter Vorbehalt lässt sich T_0 mit den angegebenen Zahlenwerten abschätzen. Bei einer Baudrate von 100 kBit/s kann T_0 bis 10 µs betragen. Sofern man den Gangfehler vernachlässigen kann (kurze Runden oder Quarzoszillator), ist ein Betrieb gerade noch möglich. Die Abschätzung ist durch den praktischen Teil dieser Arbeit sicherlich ein Stück „C16x-lastig“. Allerdings ist es nebensächlich, ob die Baudrate ohne Lagefehlerkorrektur bei 50 oder 100 kBit/s endet. Viel wichtiger ist ein Orientierungspunkt. Und hierfür eignet sich die Abschätzung in jedem Fall, da der C16x kein untypischer µC ist.

Soviel zur prinzipiellen Entstehung des „Slot-Timer“-Lagefehlers T_0. Prinzipiell deshalb, weil natürlich implementierungsspezifische Abweichungen zulässig sind. µCs mit „Auto-Reload-Timer“ können z.B. auf das explizite Nachladen verzichten, wenn der „Reload“-Wert rechtzeitig zur Verfügung steht. Die „Deadline“ liegt kurz vor dem jeweiligen „Interrupt“, da sich diese „Timer“ im Moment des „Interrupts“ selbstständig aufziehen. Aus diesem Grund kann man mit einem „Auto-Reload-Timer“ die Fehleranteile $T_{D,\,T\text{-}ISR}$ und $T_{D,\,T\text{-}Reload}$ eliminieren. Der Wermutstropfen daran ist, dass es nur wenige „low-cost“ µCs mit „Auto-Reload-Timer“ gibt. Auf das Nachladen des „Slot-Timers“ kann auch verzichtet werden, falls der erste Daten-„Slot“ für einen „Slave“ uninteressant ist. Das ändert jedoch nichts bzw. nicht viel am Lagefehler T_0. Insbesondere, wenn ein „Slave“ an mehreren „Slots“ teilnimmt. In diesem nicht seltenen Fall muss er seinen „Slot-Timer“ früher oder später nachladen. Außerdem besteht die Möglichkeit mit mehreren „Timern“ zu arbeiten, sofern sie zur Verfügung stehen. Bei einem „Slave“, der zwei „Slots“ belegt, ist das durchaus sinnvoll.

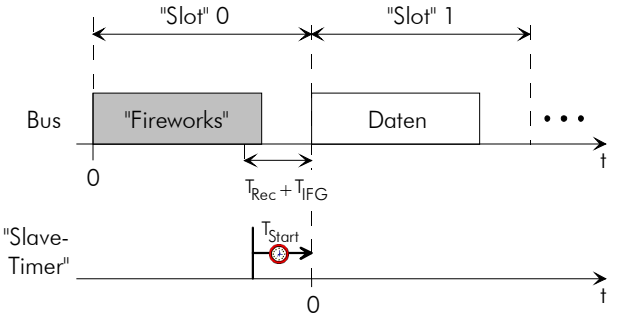
Wie erwähnt, kann auch eine UART-Emulation[84] oder ein „resetable“ UART im „Master“ für kleine Unterschiede sorgen. Die Platzierung des „Fireworks-Frames“ hat einen direkten Einfluss auf den Lagefehler in einem „Slave“. Wegen der Steuerbarkeit des Startbits, bieten die beiden Alternativen eine Möglichkeit zur Reduktion des „Fireworks“-Versatzes $T_{D,\,FW}$. Dabei besteht jedoch die Gefahr eines „Jitters“, weil diese Steuerung per Software erfolgt. Als rechen- und ressourcenstärkster Knoten arbeitet der „Master“ mit einem „großen“ µC. Deshalb muss man nicht nur mit Zeitvariationen durch „Critical-Sections“ und „Pipelines“, sondern auch durch „Caches“ und „Buffers“ rechnen. Für eine Abschätzung des „Jitters“ werden ein schneller µC mit 100 MIPS und eine Baudrate von 500 kBit/s angenommen. Außerdem soll auf dem „Master“ ein kleines RTOS laufen (längere „Critical-Sections“). Es müssen also durchaus Verzögerung von 10..15 Befehlen eingeplant werden. Auf der anderen Seite bearbeitet dieser µC 100 MIPS/500 s^{-1} = 200 Befehle pro Bit, wodurch sich der „Fireworks-Jitter“ $\Delta T_{D,\,FW}$ mit ca. $1/16 \cdot T_{Bit,\,M}$ in Grenzen hält.

Korrektur

Unter Idealbedingungen berechnet sich der „Slot-Timer“-Wert beim Nullpunktabgleich (kurz Startwert) $T_{Start,\,ideal}$ aus der IFG-Dauer T_{IFG} und dem Abstand des „Receive-Interrupt“-Zeitpunkts vom Ende eines „Frames“ T_{Rec} zu $T_{Start,\,ideal} = T_{IFG} + T_{Rec}$. T_{IFG} ist durch „Basic-TTP/A“ mit $2 \cdot T_{Bit,\,M}$ festgelegt. T_{Rec} hingegen muss man aus dem Handbuch des jeweiligen µCs bzw. UARTs ermitteln. Da die meisten UARTs den „Receive-Interrupt“ ungefähr in der Mitte des Stoppbits auslösen, hat $T_{Start,\,ideal}$ einen typischen Wert von ca. $2{,}5 \cdot T_{Bit,\,M}$. Aus den bisherigen Ergebnisse ist bekannt, dass diese Modellierung den exakten Startwert um T_0 überschätzt. Demnach muss man für eine Korrektur von $T_{Start,\,ideal}$

[84] Hiermit ist vor allem ein Software-UART gemeint. Ein Software-UART kann, muss aber nicht auf der CPU laufen. Bei den 68k µCs von Motorola eignet sich z.B. auch die „Time Processing Unit“ (siehe Abschnitt 8.3).

lediglich T_0 subtrahieren. Allerdings ist das nicht perfekt möglich, weil T_0 unbekannte Anteile enthält. Dazu zählen die zufälligen Δ-Terme in Formel 4.4 und die Signallaufzeit $T_{D,\,Sig}$; Letztere bleibt im Folgenden außen vor, da für sie bereits ein Polster von $0{,}16 \cdot T_{Bit,\,M}$ in der IFG reserviert ist. Außerdem entstehen Fehler bei der Startwerteinstellung und Bestimmung der restlichen Anteile von T_0. Deshalb bleibt, trotz Korrektur, ein Lagerestfehler $T_{0,\,Rest}$ im „Slot-Timing" übrig. Glücklicherweise sind die bekannten bzw. ermittelbaren Anteile in T_0 wesentlich größer als die unbekannten, so dass $T_{0,\,Rest}$, mit dem Korrekturansatz aus Formel 4.5, deutlich unter T_0 liegt.



$$
\begin{aligned}
T_{Start} = \; & T_{Rec} + T_{IFG} \\
& - T_{D,FW} - T_{D,UART} \\
& - T_{D,R-ISR} - T_{D,T-Load} \\
& - T_{D,T-ISR} - T_{D,T-Reload}
\end{aligned}
$$

Formel 4.5: Prinzip der „Slot-Timer"-Lagefehlerkorrektur

Der „Fireworks"-Verzug $T_{D,\,FW}$ entsteht durch den „Master". Er muss spezifiziert und den „Slaves" entweder statisch oder dynamisch (z.B. durch das IFS) bekannt gemacht werden. Die Ermittlung der „Interrupt"-Latenz- und Programmlaufzeiten ($T_{D,\,R\text{-}ISR}$, $T_{D,\,T\text{-}ISR}$, $T_{D,\,T\text{-}Load}$, $T_{D,\,T\text{-}Reload}$) ist etwas schwieriger. Hier gibt es prinzipiell eine analytische und eine messtechnische Möglichkeit. Erstere ist sehr genau, verlangt jedoch das Nachschlagen von Latenz- und Ausführungszeiten im Handbuch des jeweiligen µCs und das Auszählen von Befehlen im Maschinencode. Da diese Vorgehensweise sehr mühselig und implementierungsspezifisch ist, scheidet sie aus. Letztere ist etwas ungenauer, dafür aber implementierungsunabhängig und automatisch durchführbar (selbstjustierender Code). Eine Fehlerabschätzung wird zeigen, dass ihre Genauigkeit ausreicht. Die Messungen erfolgen mit einem „Timer" und einem instrumentierten „Basic-TTP/A"-Code[85]. Dadurch entstehen Fehler durch die Quantisierung und den zusätzlichen Code. Mit einer entsprechend hohen Auflösung lässt sich der Quantisierungsfehler klein halten. Außerdem wirkt ihm der Instrumentierungsfehler entgegen. Wenn die „Timer"-Auflösung gröber als ein Befehlszyklus ist, kann sich ein Messpunkt zwischen zwei „Timer-Ticks" befinden. Dadurch ist die gemessene Zeit zu kurz. Aufgrund des Instrumentierungscodes, wird i.A. etwas zuviel vermessen, woraus sich die Gegensinnigkeit der beiden Fehler erklärt.

Ohne implementierungsspezifische Maßnahmen werden sich die beiden Fehler selten auslöschen. Deshalb muss man einen Restfehler einkalkulieren, der anhand von drei Beispielen abgeschätzt werden soll: Im ersten Beispiel wird ein Knoten mit C16x-µC (Infineon) untersucht. Dieser µC bearbeitet einen einfachen Befehl in l00 ns (f_{CPU} = 20 MHz) – für die Messungen benötigt man i.A. nur einfache Befehle. Seine höchste „Timer"-Auflösung beträgt 200 ns. Demnach kann ein Quantisierungsfehler von -100 ns entstehen. Für den Kern der Instrumentierung benötigt man i.A. drei Befehle: „Timer" starten bzw. lesen, „Interrupt" auslösen und „Timer" stoppen bzw. lesen. Davon wirken die letzten beiden Befehle codeverlängernd, womit der Instrumentierungsfehler +200 ns beträgt. In Summe entsteht also ein Restfehler von +100 ns (ein Befehlszyklus). Das zweite Beispiel behandelt den „low-

[85] Aufgrund der einfachen Architektur von „low-cost" µCs, sind die Befehlsausführungs- und Latenzzeiten nahezu oder absolut zeitinvariant. Das erleichtert die Messungen, weil man sie unabhängig durchführen kann. Weitere Details befinden sich in Abschnitt 4.2 und im Quellcode.

cost“ µCs AT90S2313 von Atmel. Seine Befehlsdauer und höchste „Timer“-Auflösung liegen bei 100 ns (f_{CPU} = 10 MHz). Bei Verwendung des gleichen Quellcodes, beläuft sich der Restfehler auf +200 ns (zwei Befehlszyklen). Ganz ähnlich sieht es im dritten Beispiel, mit dem „low-cost“ µC PIC16F628 von Microchip, aus. Auch hier sind Befehlsdauer und höchste „Timer“-Auflösung gleich. Sie betragen 200 ns (f_{CPU} = 20 MHz), wodurch ein Restfehler von +400 ns (zwei Befehlszyklen) entsteht.

Aus den Beispielen wird ersichtlich, dass der normierte Fehler pro Messung ein bis zwei Befehlszyklen groß ist. Sofern alle vier Verzögerungen ($T_{D, R\text{-}ISR}$, $T_{D, T\text{-}ISR}$, $T_{D, T\text{-}Load}$, $T_{D, T\text{-}Reload}$) relevant sind, benötigt man für deren Bestimmung zwei Messungen – es wäre ungeschickt, würde man die aufeinanderfolgenden „Interrupt“-Latenz- und Programmlaufzeiten getrennt vermessen. Folglich beläuft sich der gesamte Messfehler auf zwei bis vier Befehlszyklen. Das entspricht in etwa einer kurzen „Critical-Section“, deren Beitrag zum Lagerestfehlers $T_{0, Rest}$, bei höheren Baudraten, mit ca. $T_{Bit, M}/16$ abgeschätzt wurde. Allerdings ist das Vorzeichen umgekehrt. Die Beispiele haben auch gezeigt, dass die Messungen eher zu groß ausfallen. Aus diesem Grund wird bei der Startwertkorrektur (siehe Formel 4.5) zuviel abgezogen. Demnach muss für die Abschätzung des Lagerestfehlers $T_{0, Rest}$ der Messfehler $\Delta T_{Mess} \approx (-1..0)/16 \cdot T_{Bit, M}$ berücksichtigt werden.

Der letzte Korrekturterm ist $T_{D, UART}$. Für seine Ermittlung muss man einen kleinen Trick anwenden. Ein „Slave“ kennt seine Bauratenabweichung gegenüber dem „Master“ mit einem Fehler von ±1,84 %. Das ist für die angestrebte Korrekturgenauigkeit zu groß, weshalb man einen anderen Weg einschlagen muss. Unter Punkt 4.1.1 wurde gezeigt, dass das „Slot-Timing“ ungefähr hundertmal genauer ist als das UART-„Timing“. Außerdem ist aus Kapitel 3 der funktionelle Zusammenhang zwischen „Slot“- und „Frame“-Dauer bekannt. Deshalb ist ein Rückschluss von der „Slot“-Dauer auf den Bittaktfehler $T_{D, UART}$ wesentlich exakter[86]. Die Berechnung ist in Formel 4.6 dargestellt und wird im Folgenden ergänzend erklärt:

$$\text{I)} \quad T_{D, UART} = m \cdot \left(T_{Bit, S} - T_{Bit, M} \right)$$

$$\text{II)} \quad T_{Bit, M} = \frac{T_{Slot, M}}{13} \approx \frac{T_{Slot, S}}{13}$$

$$\text{III)} \quad T_{Bit, S} = \frac{R_{Bit}}{f_{Osz, S}} \;,\; T_{Slot, S} = \frac{R_{Slot}}{f_{Osz, S}} \qquad \rightarrow \qquad T_{Bit, S} = T_{Slot, S} \cdot \frac{R_{Bit}}{R_{Slot}}$$

$$\Rightarrow \qquad T_{D, UART} \approx m \cdot T_{Slot, S} \cdot \left(\frac{R_{Bit}}{R_{Slot}} - \frac{1}{13} \right) \qquad \text{bzw.} \quad R_{D, UART} \approx m \cdot \left(R_{Bit} - \frac{R_{Slot}}{13} \right)$$

Formel 4.6: Ermittlung des Bittaktfehlers in einem „Slave“

Ein fehlerloser und ein baugleicher, fehlerbehafteter „Slave“ empfangen ein „Fireworks-Frame“. Der fehlerlose „Slave“ generiert $m \cdot T_{Bit, M}$ nach der Startbitflanke den „Receive-Interrupt“ (m: normierter, UART-spezifischer „Receive-Interrupt“-Zeitpunkt; typischer Wert ca. 10,5). Dasselbe passiert im fehlerbehafteten „Slave“ nach einer Zeit $m \cdot T_{Bit, S}$. Über die Differenz der beiden Werte gelangt man zur Ausgangsgleichung für die Bestimmung von $T_{D, UART}$, wobei die Subtraktion so wie in Formel 4.6

[86] Hier existiert kein Widerspruch, da die „Slot“-Dauer vom „Slot-Timer“-Lagefehler unabhängig ist. Erstere ist entweder a priori (Quarzoszillator) oder aus der Gangfehlerkorrektur bekannt.

durchgeführt werden muss, damit das Vorzeichen gemäß Formel 4.5 stimmt. In der Ausgangsgleichung sind die zwei Unbekannten $T_{Bit, M}$ und $T_{Bit, S}$ enthalten. Für die Eliminierung der „Master"-Bitzeit $T_{Bit, M}$ wird der erwähnte Ansatz $T_{Slot, S} \approx T_{Slot, M}$ herangezogen. Die „Slave"-Bitzeit $T_{Bit, S}$ ist insofern unbekannt, weil ein „Slave" seine Oszillatorfrequenz $f_{Osz, S}$ nicht exakt kennt. Aus diesem Grund muss $T_{Bit, S}$ über ein Verhältnis von Registerwerte ausgedrückt werden. In Formel 4.6 steht R_{Bit} für den Wert des Baudratenregisters und R_{Slot} für den „Slot-Dauer"-Wert des „Timer"-Registers – die Formeln können von µC zu µC ein wenig variieren.

Angesichts der Fehlergrenzen für das „Slot-Timing" von etwa ±150 ppm, kann der Schätzfehler von $T_{D, UART}$ ($\approx 10{,}5 \cdot T_{Bit, M} \cdot 150 \cdot 10^{-6} \approx 1{,}58 \cdot 10^{-3} \cdot T_{Bit, M}$) getrost vernachlässigt werden. Eine Folge dieser Methode ist die stete Aktualisierung von $T_{D, UART}$ bei „Slaves" mit ungenauen RC- oder „On-Chip"-Oszillatoren. Diese „Slaves" benötigen i.A. eine Gangfehlerkorrektur und UART-Synchronisation (siehe Abschnitt 4.1.3 und 4.1.4), wodurch sich die Werte R_{Slot} und R_{Bit} ändern können. Ein günstiger Zeitpunkt hierfür ist die IRG (siehe Abschnitt 4.2). Bei „Slaves" mit genauen Oszillatoren reicht dagegen eine einmalige A-Priori-Berechnung aus, sofern die Baudrate nicht genau einstellbar ist[87].

Damit sind einem „Slave" alle Größen für die Berechnung des korrigierten Startwerts T_{Start} bekannt. Entgegen Formel 4.5, wo das Prinzip der Lagefehlerkorrektur gezeigt ist, macht es Sinn, den Startwert T_{Start} zu runden. Theoretisch ist T_{Start} zwar sehr genau bekannt, praktisch kann dieser Wert, wegen der Quantisierung, in den meisten Fällen aber nicht exakt eingestellt werden. Mit einer Rundung[88], so wie in Formel 4.7 dargestellt, bewirkt man, dass sich der Quantisierungsfehler ΔT_S symmetrisch um den Nominalwert verteilt ($T_{Start} \pm ½ \cdot \Delta T_S$).

$$T_{Start} = \mathrm{rnd}\left(T_{Rec} + T_{IFG} - T_{D, FW} - T_{D, UART} - \sum T_{Mess} \right)$$

Formel 4.7: Berechnung des korrigierten Startwerts

Restfehler

Eine Restfehlerbetrachtung soll diesen Unterpunkt abschließen. Die unsystematischen Fehler (Δ-Werte) können, aufgrund ihrer unbekannten, zeitlichen Änderung, nicht in die Lagefehlerkorrektur einfließen. Im Speziellen handelt es sich um den möglichen „Fireworks-Jitter" $\Delta T_{D, FW}$ bei einer UART-Emulation oder einem „resetable" UART im „Master", die Unsicherheit bei der Startbiterkennung $\Delta T_{D, UART}$, zusätzlicher „Interrupt"-Verzögerungen $\Delta T_{D, ISR}$ (vor allem wegen „Critical-Sections"), Ungenauigkeiten bei der Codevermessung ΔT_{Mess} und Einstellfehlern durch die „Timer"-Quantisierung ΔT_S. Gemäß den vorangehenden Abschätzungen und Berechnungen betragen diese zufälligen Fehler:

- $\Delta T_{D, FW} \approx 0..T_{Bit, M}/16$
- $\Delta T_{D, UART} \approx 0..T_{Bit, M}/8$ (achtfache Bitteilung)
- $\Delta T_{D, ISR} \approx 0..T_{Bit, M}/8$ (zwei „Interrupts")
- $\Delta T_{Mess} \approx -T_{Bit, M}/16..0$
- $\Delta T_S \approx \pm T_{Bit, M}/16$. (min. „Slot-Timer"-Auflösung)

[87] Das ist jedoch eher die Ausnahme, weil man beim Design eines Knotens die Oszillatorfrequenz auf die gewünschte Baudrate abstimmt.

[88] Da die Rundungsoperation in den Bibliotheken oft langsam ist, sollte man sie selbst programmieren. Einfach, klein und schnell ist folgende Integer-Variante: $x^* = \lfloor SHR(SHL(x)+1) \rfloor$ mit SHR = Multipl. mit ½ und SHL = Multipl. mit 2.

Wenn man vom „Worst-Case“ ausgeht, so bewegt sich der Lagerestfehler des „Slot-Timings“ $T_{0, Rest}$ zwischen etwa (-2..6)/16·$T_{Bit, M}$. Das heißt, dass für die Kompensation des Lagerestfehler $T_{0, Rest}$ die Hälfte der IFG-Restdauer ($T_{Bit, M}$) benötigt wird. An und für sich stellt das kein Problem dar, weil der Rest von ≈ ½·$T_{Bit, M}$ für die Kompensation des Gangfehlers theoretisch ausreicht. Die „Worst-Case“-Abschätzung ist jedoch extrem pessimistisch. Erstens wird der eher seltene „Fireworks-Jitter“ $\Delta T_{D, FW}$ berücksichtigt. Zweitens wird von einem „Slave“ mit achtfacher Bitabtastung ausgegangen (typisch ist eine sechszehnfache Bitabtastung). Drittens werden beim Nullpunktabgleich zwei „Interrupts“ angenommen, was nicht zwingend ist (siehe Abschnitt 4.2.5). Viertens wird von der min. „Slot-Timer“-Auflösung $\Delta T_{S, max} = 1/8 \cdot T_{Bit, M}$ ausgegangen und fünftes von der gleichgerichteten, maximalen Überlagerung der Fehler.

Aus diesen Gründen ist der „Worst-Case“ sehr unwahrscheinlich, weshalb eine statistische Abschätzung des Lagerestfehler $T_{0, Rest}$ angebracht ist. Dafür wird eine gleichmäßige Verteilung der Fehler um den jeweiligen Mittelwert angenommen, womit die halben Fehlerspannweiten zu Streuungen σ_i werden. Die Berechnung der Gesamtstreuung σ erfolgt über die Summe der Varianzen σ_i^2: $\sigma^2 = \Sigma\sigma_i^2$. Mit den obigen Zahlwerten folgt daraus: $\sigma = T_{Bit, M} \cdot (1/32^2 + 1/16^2 + 1/16^2 + 1/32^2 + 1/16^2)^{0,5} \approx 2/16\ T_{Bit, M}$. Der Rückschluss $T_{0, Rest} \approx \pm\sigma$ liefert eine Restfehlerspannweite von etwa $0{,}25 \cdot T_{Bit, M}$. Zuzüglich einer kleinen Reserve, lässt sich für $T_{0, Rest}$ schließlich eine Spannweite von $\approx 0{,}3 \cdot T_{Bit, M}$ angeben.

4.1.3 „Slot-Timer“-Gangfehlerkorrektur

Ursachen

Für den „Slot-Timer“-Gangfehler ist der Unterschied zwischen den „Slot“-Perioden in einem „Slave“ und im „Master“ ($T_{Slot, S} \neq T_{Slot, M}$) verantwortlich. Diese Ungleichheit kann prinzipiell drei Ursachen haben. Nach Formel 4.6, ergibt sich die „Slot“-Periode in einem Knoten aus der Oszillatorfrequenz f_{Osz} und dem Registerwert für den „Slot-Timer“ R_{Slot} zu $T_{Slot} = R_{Slot}/f_{Osz}$. Die Oszillatorfrequenz ist weder im „Master“, noch in einem „Slave“ konstant. Für sie gilt $f_{Osz} = f_{Osz, Norm} + \Delta f_{Osz}(t)$. Hierbei ist $f_{Osz, Norm}$ die Nominalfrequenz und $\Delta f_{Osz}(t)$ der Frequenzfehler. $\Delta f_{Osz}(t)$ ist eine Funktion der Zeit, da ein Oszillator, neben dem konstanten Abgleichfehler, gegen Temperatur- und Betriebsspannungsschwankungen empfindlich ist und mit zunehmendem Alter seine Frequenz ändert – von allen variablen Anteilen ist der Temperaturdurchgriff am größten. Ersetzt man in der Gleichung für die „Slot“-Periode f_{Osz} durch $f_{Osz, Norm}$ und $\Delta f_{Osz}(t)$, und nimmt man des Weiteren einmal $T_{Slot, S} = T_{Slot, M}$ an, so werden die drei prinzipiellen Ursachen für den „Slot-Timer“-Gangfehler offensichtlich:

$$\frac{f_{Osz, Norm, S}}{R_{Slot, S}} + \frac{\Delta f_{Osz, S}(t)}{R_{Slot, S}} \stackrel{!}{=} \frac{f_{Osz, Norm, M}}{R_{Slot, M}} + \frac{\Delta f_{Osz, M}(t)}{R_{Slot, M}}$$

$$\Rightarrow \quad \frac{\Delta f_{Osz, S}(t)}{R_{Slot, S}} = \frac{\Delta f_{Osz, M}(t)}{R_{Slot, M}} \quad \wedge \quad \frac{f_{Osz, Norm, S}}{R_{Slot, S}} = \frac{f_{Osz, Norm, M}}{R_{Slot, M}}$$

Aufgrund der unsystematischen Oszillatorfehler im „Master“ und in einem „Slave“ (zwei Ursachen), ist die erste Bedingung, die Gleichheit der variablen Terme, i.A. nicht erfüllbar. Dagegen kann die zweite Bedingung, die Gleichheit der konstanten Terme, schon erfüllt werden. Allerdings ist das nicht selbstverständlich. Bei einer unglücklichen Paarung der Oszillatorfrequenz $f_{Osz, Norm, S}$ und des „Slot-Timer“-Registerwerts $R_{Slot, S}$, entsteht in einem „Slave“ ein zusätzlicher Fehler (dritte Ursache). Da es

sich um einen systematischen Fehler handelt, kann er vermieden werden. Aus diesem Grund wird die bedachte Dimensionierung des Wertepaars $f_{Osz, Norm, S}$ und $R_{Slot, S}$ beim Entwurf eines „Slaves“ gefordert, was im Normalfall kein Problem darstellt. Damit lässt sich der „Slot-Timer“-Gangfehler auf zwei Ursachen reduzieren:

$$\Delta f_{Slot, S}(t) = \frac{\Delta f_{Osz, M}(t)}{R_{Slot, M}} - \frac{\Delta f_{Osz, S}(t)}{R_{Slot, S}}$$

Formel 4.8: „Slot-Timer“-Gangfehler

Die Notwendigkeit für die Eliminierung des systematischen Fehlers soll folgendes Beispiel erläutern: Es werden Knoten („Master“ und „Slave“) mit Standardquarzoszillatoren angenommen. Aus Kostengründen sollen die Oszillatoren nicht feinjustiert sein, so dass mit einem Abgleichfehler von ca. $\pm 50 \cdot 10^{-6}$ und einen Temperaturfehler von ca. $\pm 30 \cdot 10^{-6}$ (υ = -20..70°C) gerechnet werden muss (siehe Abschnitt 8.4). Gemäß Formel 4.8, gilt somit für den „Slot-Timer“-Gangfehler eines „Slaves“ $|\Delta f_{Osz, S}|_{max} = 160 \cdot 10^{-6}$. Aus Abschnitt 4.1.1 ist bekannt, dass der Betragsfehler für das „Slot-Timing“ unter widrigen Umständen $152 \cdot 10^{-6}$ nicht überschreiten darf. Des Weiteren ist mit den Ergebnissen aus Abschnitt 4.1.2 der Anteil des „Slot-Timer“-Lagerestfehlers berechenbar. Er beläuft sich auf ungefähr $(\pm 2/16 \cdot T_{Bit, M})/(253 \cdot 13 \cdot T_{Bit, M}) \approx \pm 38 \cdot 10^{-6}$ und reduziert das Toleranzband für den „Slot-Timer“-Gangfehler. Man darf jedoch davon ausgehen, dass bei einem wohlbedachten Entwurf und der Verwendung von Quarzoszillatoren, der Längenfehler der UART-„Frames“ weit unter seiner zulässigen Grenze liegt. Dadurch wird bei quarzbetriebenen Knoten ein Stück des UART-Toleranzbandes für anderweitige Zeitfehler frei. Die Größe dieses Freiraums liegt bei etwa $0{,}375 \cdot T_{Bit, M}$ (Schnitt nach ungefähr $10{,}625 \cdot T_{Bit, M}$). Für das relative Toleranzband des „Slot-Timer“-Gangfehlers bedeutet dies eine Vergrößerung um ca. $(\pm ½ \cdot 0{,}375 \cdot T_{Bit, M})/(253 \cdot 13 \cdot T_{Bit, M}) \approx \pm 57 \cdot 10^{-6}$, wodurch es auf etwa $\pm 171 \cdot 10^{-6}$ anwächst. Das reicht für einen Betrieb von „Slaves“ mit Standardquarzoszillatoren ohne Gangfehlerkorrektur gerade aus.

Ohne die Eliminierung des systematischen Fehlers wäre das, zumindest theoretisch, nicht möglich. Denn die Reserve von $\pm 10 \cdot 10^{-6}$ ist schnell aufgebraucht. Man muss jedoch dazusagen, dass es sich hier um einen Extremfall handelt. Schließlich sind Runden mit voller Länge genauso selten wie Temperaturunterschiede von 90 K und maximale, gleichgerichtete Fehler. Deshalb sind die Reserven in der Praxis sehr viel größer. Im Gegensatz dazu benötigen „Slaves“ mit günstigeren, aber ungenaueren Oszillatoren in vielen Fällen eine Gangfehlerkorrektur. Angesichts des vorangehenden Beispiels mag es verwundern, dass das für ungenauere Oszillatoren nicht generell gilt. Man muss jedoch die Vielfalt der Faktoren bedenken. Dazu zwei Beispiele: Bei RC- oder „On-Chip“-Oszillatoren, deren typischer Abgleich- und Temperaturfehler ohne Trimmung bei grob ±(5..10) % liegt (siehe Abschnitt 8.4), ist eine Gangfehlerkorrektur erforderlich. Ein Keramikoszillator weist hingegen einen Abgleich- und Temperaturfehler von unter ±1 % auf. Wenn ein damit ausgestatteter „Slave“ an den hinteren „Slots“ einer langen Runde teilnimmt, benötigt er ebenfalls eine Gangfehlerkorrektur. Sollten für ihn allerdings nur die vorderen „Slots“ interessant sein, kann er darauf verzichten. Denn nach Formel 4.3 spielt der Gangfehler nur bei längerfristigen Vorhersagen eine Rolle.

Korrektur

Die Gangfehlerkorrektur orientiert sich an den Möglichkeiten von gewöhnlichen „low-cost“ µCs. Es wird ein „Timer“, ein „Receive-Interrupt“, ein bisschen Speicher und etwas Rechenzeit benötigt. Sofern ein µC keinen zweiten „Timer“ besitzt, kann man auch den „Slot-Timer“ verwenden. Ein Hard-

ware-UART ist ebenso wenig erforderlich. Die Gangfehlerkorrektur arbeitet genauso gut mit einer UART-Emulation (siehe Abschnitt 8.3). Aufgrund dieser minimalen Voraussetzungen, ist die Gangfehlerkorrektur prinzipiell auf jedem „Slave“ lauffähig. Des Weiteren wird von einer quasistationären Oszillatorfrequenz ausgegangen. In Abschnitt 4.1.1 wurde gezeigt, dass die zeitliche Änderung der Oszillatorfrequenz innerhalb „Basic-TTP/A“-typischer Zeiträume vernachlässigbar ist. Folglich darf auch hier mit der Approximation $\Delta f_{Slot,\,S}(t) \approx \Delta f_{Slot,\,S} = \text{konst.}$ gearbeitet werden. Das vereinfacht zum einen den Korrekturalgorithmus und sorgt zum anderen für eine schwache Realzeitbedingung. Letzteres gestattet die Durchführung der Gangfehlerkorrektur in einer niederprioren „Task“, wodurch keine Konflikte mit den zeitkritischen „Basic-TTP/A“-Vorgängen entstehen können.

Das Prinzip der Gangfehlerkorrektur ist in Bild 4.5 dargestellt. Es beruht auf dem Vermessen der Zeitintervalle zwischen zwei „Fireworks-Frames“ und einer Extrapolation für die „Slot“-Vorhersage. Angesichts der hohen Genauigkeit des „Slot-Timings“, sind die „Fireworks-Frames“ die einzigen „Frames“, die sich als implizite Zeitmarken zum Vermessen eignen. Sie stammen erstens immer vom „Master“ und sind zweitens „Jitter“-frei bzw. „Jitter“-arm. Dadurch ist sichergestellt, dass sich ein „Slave“ an der Zeitreferenz orientiert und die Messfehler gering ausfallen. Für die Messungen wird der erwähnte „Timer“ gebraucht. Ein separater „Timer“ bietet den Vorteil, die Zeiträume zwischen den „Fireworks-Frames“ direkt messen zu können, sofern sein Modulo ausreicht. Allerdings verfügen einige „low-cost“ µCs nur über einen „Timer“. In diesen Fällen muss man die Messungen mit dem „Slot-Timer“ indirekt durchführen. Der „Slot-Timer“ wird i.A. nachgeladen, wodurch die Summierung der einzelnen „Slot-Timer“-Zeiten erforderlich ist.

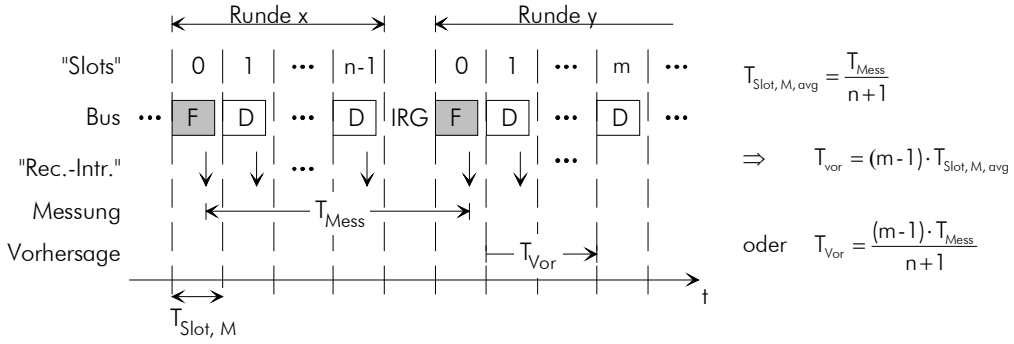


Bild 4.5: Prinzip der Gangfehlerkorrektur

Nach einer Messung berechnet ein „Slave“ die, auf seine Zeitbasis bezogene, gemittelte „Master-Slot“-Periode $T_{Slot,\,M,\,avg}$. In dem Beispiel in Bild 4.5 teilt er dazu die gemessene Zeit T_{Mess} durch die Anzahl der „Slots“ n in der Runde x, zuzüglich einem „Slot“ für die IRG[89]. Je nach Messdauer und Rundenlängen, kann sich eine Messung natürlich auch über mehrere Runden und IRGs erstrecken. Eine lineare Extrapolation, mit $T_{Slot,\,M,\,avg}$ als Steigung, erlaubt einem „Slave“, aufgrund der quasistationären Oszillatorfrequenzen („Master“ und „Slave“), eine ausreichend genaue „Slot“-Vorhersage. Allerdings ist der Vorsagezeitraum T_{Vor} nicht beliebig groß, da $T_{Slot,\,M,\,avg}$ Fehler enthält. Diese werden jedoch umso kleiner, je länger man misst. Im Vorgriff sei erwähnt, dass die min. Messdauer ungefähr der max. Vorhersagezeit entspricht[90]. Für die Kompensation des Frequenzdrifts $\Delta f_{Slot,\,S}(t)$ muss die Messung regelmäßig wiederholt werden. Das Intervall hängt vom Einsatzort und den thermischen Zeitkonstanten des jeweiligen „Slaves“ ab. Letztere bewegen sich typischerweise im oberen s-Bereich,

[89] Die Anzahl der „Slots“ pro Runde kennt ein „Slave“ aus seiner RODL (siehe Abschnitt 3.2.3).

[90] Hierin ist die Gesetzmäßigkeit „Messzeit ist Genauigkeit“ enthalten.

weshalb eine Wiederholung alle 2..3 s angebracht ist. Damit fällt die relative Rechenlast der Gangfehlerkorrektur gering aus, wodurch die Berechnungen, Korrektur- und Messvorbereitungen i.A. genug Zeit haben[91]. Zeitkritisch sind hingegen die Messung und Korrektur selbst. Sie werden, angestoßen durch die Gangfehlerkorrektur-„Task“, in der „Receive-“ und/oder „Slot-Timer-Task“ durchgeführt (siehe Abschnitt 4.2.6).

Restfehler

Es wurde bereits angedeutet, dass die Gangfehlerkorrektur nicht perfekt ist, wodurch ein kleines $\Delta f_{Slot,\,S}$ übrig bleibt. Dieser Gangrestfehler entsteht zum einen durch zufällige Abweichungen bei der Messung. Dazu zählen die „Timer“-Auflösung ΔT_M und die aus Abschnitt 4.1.2 bekannten Zeitunsicherheiten beim Empfang eines „Fireworks-Frames“ („Fireworks-Jitter“ $\Delta T_{D,\,FW}$, „Interrupt“-Verzögerungen $\Delta T_{D,\,ISR}$, Startbiterkennung $\Delta T_{D,\,UART}$). Zum anderen haben Einstellfehler der Vorhersagezeit T_{Vor}, wegen der begrenzten „Slot-Timer“-Auflösung ΔT_S, einen Anteil am Gangrestfehler. Die Zahl der Abweichungen lässt jedoch erahnen, dass der Messfehler die weitaus größere Rolle spielt. Konstante und quasikonstante Fehler wie die Baudraten-Abweichung, „Interrupt“-Latenzzeiten, Programmlaufzeiten usw., haben keinen Einfluss. Sie werden durch die Messtechnik, bei der nur Differenzen zählen, automatisch eliminiert.

Für die Abschätzung des Gangrestfehlers, wird aus Gründen der Übersichtlichkeit in einem ersten Schritt der Messfehler ermittelt. Den wesentlichen Anteil bilden die oben genannten Zeitunsicherheiten beim Vermessen des Abstandes zwischen zwei „Fireworks-Frames“. Daran sind nicht nur ihre Anzahl und Größe schuld, sondern auch eine Spannweitenverdopplung. Letzteres entsteht durch die Relativmessungen und soll kurz erläutert werden. Zwei „Fireworks-Frames“ treten zu den Absolutzeiten $t_1 + \Delta T(t_1)$ und $t_2 + \Delta T(t_2)$ auf; $\Delta T(t)$ ist eine zufällige Verzögerung. Folglich ist der Abstand zwischen diesen „Fireworks-Frames“: $T_{FW} = t_1 - t_2 + \Delta T(t_1) - \Delta T(t_2)$. Wenn $\Delta T(t_1)$ Null und $\Delta T(t_2)$ maximal groß ist, weicht T_{FW} um $-\Delta T_{max}$ vom Sollwert $t_1 - t_2$ ab. Im umgekehrten Fall beträgt die resultierende Abweichung $+\Delta T_{max}$. Deshalb gilt für den möglichen Fehler des Abstands zwischen zwei „Fireworks-Frames“ $\pm\Delta T_{max}$. Gemäß Abschnitt 4.1.2, können bei der „Fireworks“-Erkennung folgende Fehler entstehen:

- $\Delta T_{D,\,FW} \approx 0..T_{Bit,\,M}/16$
- $\Delta T_{D,\,UART} \approx 0..T_{Bit,\,M}/8$
- $\Delta T_{D,\,ISR} \approx 0..T_{Bit,\,M}/8$

Bei einer Summierung der Maximalwerte kann der Messfehleranteil theoretisch $\pm 5/16 \cdot T_{Bit,\,M}$ groß werden. Allerdings wurde in Abschnitt 4.1.2 auch beschrieben, dass dieser „Worst-Case“ extrem pessimistisch und deshalb eine Abschätzung über die Varianzen angebracht ist. Als realistischer Messfehleranteil wird somit festgelegt: $\Delta T_{Mess} \approx \pm(1/16^2 + 1/8^2 + 1/8^2)^{0,5} \cdot T_{Bit,\,M} = \pm 3/16 \cdot T_{Bit,\,M}$.

Einen weiteren Beitrag liefert die Granularität ΔT_M des Mess-„Timers“. Während einer Messung registriert er $k = \lfloor T_{Mess}/\Delta T_M \rfloor$ „Ticks“. Durch die Abrundung ist der Rückschluss $T_{Mess} = k \cdot \Delta T_M$ in der Regel nicht exakt, da der wahre Wert T_{Mess} irgendwo im halboffenen Intervall $[k \cdot \Delta T_M;\ (k+1) \cdot \Delta T_M[$ liegen kann. Folglich beträgt der gesamte Messfehler $\Delta T_{Mess,\,ges} \approx -3/16 \cdot T_{Bit,\,M}..+3/16 \cdot T_{Bit,\,M} + \Delta T_M$. Aus Vereinfachungsgründen wird dieser Bereich symmetriert. Das ändert die Zahlenwerte ein bisschen, jedoch nichts am Ergebnis. Schließlich steht die notwendige Kompensationszeit in der IFG für den Gangrestfehler im Vordergrund. Und dafür zählt nach wie vor nur die Fehlerspannweite. Aus diesem Grund ist als Gesamtmessfehler die Angabe $\Delta T_{Mess,\,ges} \approx \pm(3/16 \cdot T_{Bit,\,M} + ½ \cdot \Delta T_M)$ zulässig.

[91] Zum Vergleich: Eine max. lange Runde mit 255 „Slots“ dauert bei einer sehr niedrigen Baudrate von 9,6 kBit/s ca. 0,35 s.

Soviel zum Messfehler, der einen Teil im Gangrestfehler darstellt. Den anderen Teil bildet der Einstellfehler des „Slot-Timers“, aufgrund seiner begrenzten Auflösung. Es wird nur in den seltensten Fällen passieren, dass die Vorhersagezeit T_{Vor} ganzzahlig durch die „Slot-Timer“-Auflösung ΔT_S teilbar ist. Deshalb muss bei der „Slot“-Vorhersage ein Einstellfehler mit einer Spannweite von bis zu ΔT_S berücksichtigt werden. Über eine Rundung lässt sich der Einstellfehler gleichmäßig um den Sollwert verteilen, so dass $\Delta T_{Quant} = \pm ½ \cdot \Delta T_S$ ist. Formel 4.9 zeigt die entsprechende Rechenvorschrift. Sie ist aus dem Prinzip in Bild 4.5 abgeleitet und enthält eine empfehlenswerte „Integer“-Umsetzung des Rundungsoperators rnd() – der Schneideoperator $\lfloor\rfloor$ muss nicht programmiert werden. Bis auf m und i, wurden alle Variablen ausführlich erklärt. $m \in \{1..254\}$ ist die Nummer des zu vorhersagenden Daten-„Slots“ und i die Anzahl der „Slots“ in einer vorangegangen Messung.

$$T_{Vor}(m) = \mathrm{rnd}\left(\frac{m-1}{i} \cdot \frac{T_{Mess}}{\Delta T_S}\right) \cdot \Delta T_S = \left\lfloor \frac{1}{2}\left(2 \cdot \frac{m-1}{i} \cdot \frac{T_{Mess}}{\Delta T_S} + 1\right)\right\rfloor \cdot \Delta T_S$$

Formel 4.9: Rechenvorschrift für die „Slot“-Vorhersage

Nachdem die Rechenvorschrift für die „Slot“-Vorhersage bekannt ist und die Teilfehler quantifiziert sind, kann der Gesamtfehler abgeschätzt werden. Hierfür wird das totale Differential herangezogen, weil es einen funktionalen Zusammenhang zwischen dem Gesamtfehler und seinen Ursachen liefert. Im Anschluss wird daraus eine Dimensionierungsregel für die Messdauer abgeleitet.

$$\left|\Delta T_{Vor}\right|_{max} \approx \left|\frac{\partial T_{Vor}}{\partial T_{Mess}} \cdot \Delta T_{Mess,\,max}\right| + \left|\frac{\partial T_{Vor}}{\partial T_{Vor}} \cdot \Delta T_{Quant,\,max}\right|$$

$$\frac{\partial T_{Vor}}{\partial T_{Mess}} = \frac{m-1}{i} \qquad \Delta T_{Mess,\,max} = \frac{3}{16} \cdot T_{Bit,\,M} + \frac{\Delta T_M}{2}$$

$$\frac{\partial T_{Vor}}{\partial T_{Vor}} = 1 \qquad \Delta T_{Quant,\,max} = \frac{\Delta T_S}{2}$$

$$\Rightarrow \qquad \left|\Delta T_{Vor}\right|_{max} \approx \frac{m-1}{i} \cdot \left(\frac{3}{16} \cdot T_{Bit,\,M} + \frac{\Delta T_M}{2}\right) + \frac{\Delta T_S}{2}$$

Formel 4.10: Vorhersagefehler eines „Slots“

Anmerkung: Unter formaler Beachtung der Kettenregel und der Rundungsfunktion, stünde in Formel 4.10 nur der Term $d(rnd(x))/dx \cdot dx/dT_{Mess}$. Die Unstetigkeit der Rundungsfunktion würde jedoch zu Distributionen führen, die für eine Dimensionierungsregel wenig vorteilhaft wären. Aus diesem Grund wurde ein Kunstgriff angewandt, der den Rundungsfehler über die Empfindlichkeit von T_{Vor} gegenüber sich selbst berücksichtigt.

Aus dem Ergebnis geht hervor, dass der max. Vorhersagefehler ($|\Delta T_{Vor}|_{max}$) umso kleiner wird, je länger die Messdauer (i) ist und je höher die „Timer“-Auflösungen (ΔT_S, ΔT_M) sind. Über diese Steu-

erbarkeit kann man den max. Vorhersagefehler kleiner machen als die noch zur Verfügung stehende Kompensationszeit von ca. $0{,}7 \cdot T_{Bit,\,M}$ in der IFG (siehe Abschnitt 4.1.2). Nun ist die Gangfehlerkorrektur i.A. für „Slaves“ mit ungenauen „low-cost“ Oszillatoren und „low-cost“ µCs interessant. Da Letztere oftmals nur einen „Timer“ besitzen, wird automatisch $\Delta T_M = \Delta T_S$. Außerdem ist die „Timer“-Auflösung von einigen „low-cost“ µCs bei hohen Baudraten nicht größer als $T_{Bit,\,M}/8$ – das sind in der Regel die µCs mit achtfach abtastendem UART, wie z.B. der ST62x18. Dadurch wird in Formel 4.10 $\Delta T_M = \Delta T_S = \pm T_{Bit,\,M}/16$, entsprechend den Ausführungen zuvor. Als Steuerparameter verbleibt die Anzahl i der „Slots“ während einer Messung. Nach Umstellen von Formel 4.10 erhält man folgende Dimensionierungsregel:

$$\frac{0{,}7}{2} \cdot T_{Bit,\,M} \geq \frac{m-1}{i} \cdot \left(\frac{3}{16} \cdot T_{Bit,\,M} + \frac{1}{16} \cdot T_{Bit,\,M} \right) + \frac{1}{16} \cdot T_{Bit,\,M}$$

$$\Rightarrow \qquad i \geq 0{,}87 \cdot (m-1)$$

Formel 4.11: Dimensionierungsregel für die Messdauer

Eine Auswertung von Formel 4.11 liefert für den Extremfall m = 254 einen Minimalwert für i von 220,11. Sollte ein „Slave“ also im letzten „Slot“ einer max. langen Runde kommunizieren wollen, muss er zuvor min. 221 „Slots“ vermessen haben. Die Mindestdauer dieser Messung hängt von der Baudrate ab. Bei einer hohen Baudrate von z.B. 500 kBit/s würde sie knapp 6 ms betragen und bei einer niedrigen Baudrate von z.B. 9,6 kBit/s ca. 0,3 s. Im Vergleich zur Messperiode von ca. 2..3 s, ist jedoch auch die „Worst-Case“-Messdauer kurz, weshalb die Gangfehlerkorrektur ein zeitunkritischer Vorgang ist.

Die Gültigkeit von Formel 4.11 soll anhand der Beispiele in Bild 4.6 unterlegt werden. Es sind die oberen und unteren Grenzkurven der Vorhersagefehler über dem Vorhersagezeitraum m, für verschiedene Parameter i und ΔT_S, aufgetragen. Dazu wurden Formel 4.9 rechnergestützt ausgewertet und die Ergebnisse mit den Sollwerten verglichen. Im Diagramm oben, links ist i = 50 und $\Delta T_S = T_{Bit,\,M}/8$. Wie erwartet, ist der Vorhersagezeitraum – Schnittpunkte der Kurven mit den Ordinaten ±0,35 – nicht wesentlich größer als i. Dass die ersten „Slots“ keine Gangfehler aufweisen, liegt am ganzzahligen Verhältnis $T_{Bit,\,M}/\Delta T_S$. Aus dem gleichen Grund sind die Kurven auch treppenförmig und symmetrisch. Wenn dieses Verhältnis nicht geradzahlig ist, tritt der Gangfehler sofort in Erscheinung. Das zeigt das nächste Diagramm oben, rechts mit $T_{Bit,\,M}/\Delta T_S = 7{,}965$ und i = 100. Der doppelte, sägezahnförmige Verlauf entsteht wegen der Schneideoperation beim Messen und der Rundungsoperation beim Einstellen. Außerdem ist hier der Messfehler ΔT_{Mess} nicht ganzzahlig durch ΔT_S teilbar, weshalb die Kurven unsymmetrisch zueinander liegen – das gilt auch, wenn $T_{Bit,\,M}/\Delta T_S$ ganzzahlig ist.

Die beiden unteren Diagramme zeigen die Gangfehlergrenzen bei feineren „Timer“-Auflösungen ($\Delta T_M = \Delta T_S$) als dem Minimum von $T_{Bit,\,M}/8$. Man erkennt, insbesondere im linken Diagramm mit $T_{Bit,\,M}/\Delta T_S = 16{,}029$, dass die Vorhersagegrenze ein gutes Stück über dem Parameterwert i liegt. Aus diesem Grund sollten die „Timer“-Auflösungen, mit Rücksicht auf die Ressourcen eines „Slaves“, möglichst hoch gewählt werden. Des Weiteren ist das rechte Diagramm ein Beispiel für die Vielfalt der Kurvenformen. Einen entscheidenden Einfluss darauf hat das Verhältnis $T_{Bit,\,M}/\Delta T_S$. An bestimmten Punkten reichen sehr kleine Änderungen von $T_{Bit,\,M}/\Delta T_S$ aus, um ein völlig anderes Fehlerbild zu erzeugen. Bei einer rein messtechnischen Betrachtung kann dieses tückische Verhalten leicht zu einer Unterschätzung des Gangfehlers führen. Das gilt vor allem für die untere Grenzkurve. Sie reagiert

empfindlicher auf Veränderungen von $T_{Bit, M}/\Delta T_S$ als die Obere. Der Grund ist die Schneideoperation beim Messen, wodurch die gemessene Zeit in der Regel zu kurz ausfällt. Als Folge trifft die untere Kurve i.A. zuerst auf ihren Grenzwert, was in den beiden rechten Diagrammen schön zu sehen ist. Deshalb sollte man zumindest, die aus Formel 4.11 ableitbare, einfache Faustregel „Messzeit ≈ längste Vorhersagezeit“ befolgen[92].

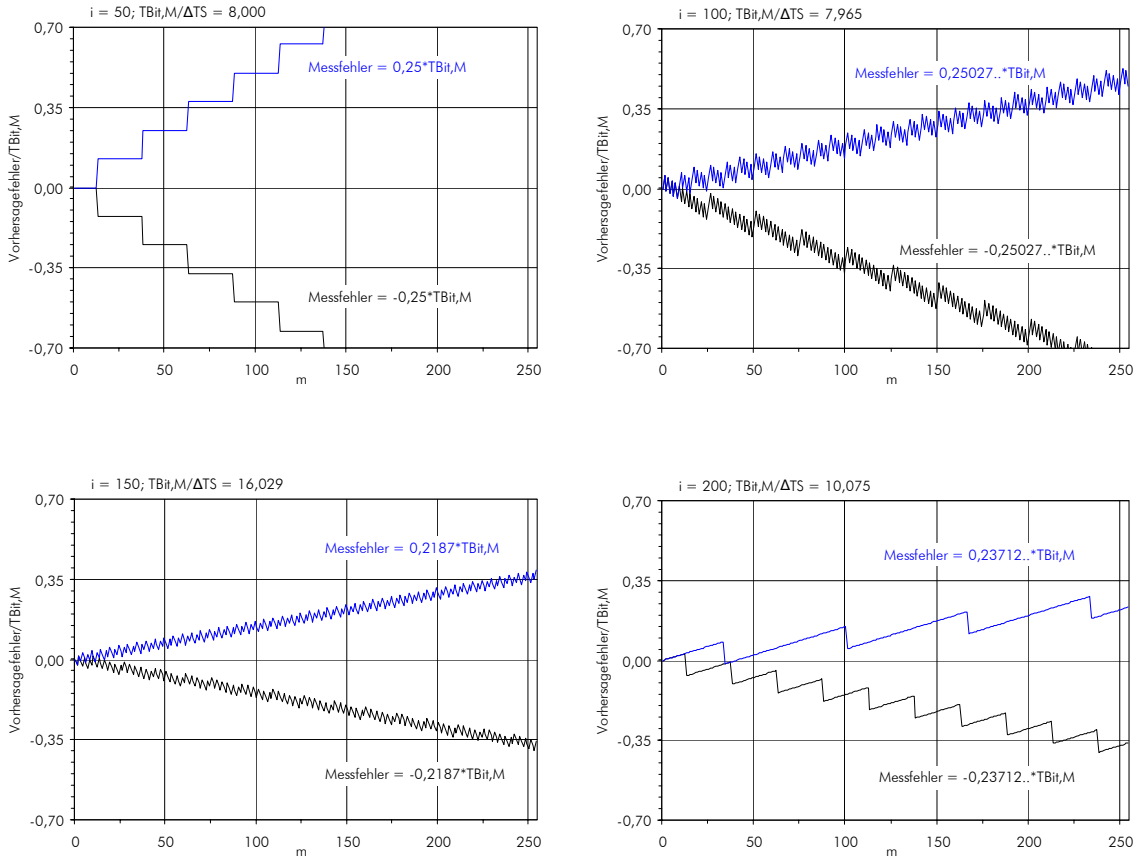


Bild 4.6: Beispiele für den Vorhersagefehler

Abschließend ein paar Bemerkungen für eine Implementierung: Um eine Fehlerfortpflanzung bei der „Slot“-Vorhersage zu vermeiden, sollte man in Formel 4.9 nur eine Division durchführen. In Ausnahmenfällen, wenn ein „Slave“ an vielen „Slots“ in einer Runde teilnimmt und vielleicht sogar über einen „Auto-Reload-Timer“ verfügt, kann die „Master-Slot“-Periode $T_{Slot, M}$ auch separat berechnet werden. Allerdings muss man zusätzliche Maßnahmen für die Kontrolle der Fehlerfortpflanzung treffen. Durch die separate Berechnung der „Master-Slot“-Periode $T_{Slot, M} = rnd(T_{Mess}/i)$ entsteht pro „Slot“ ein Rundungsfehler $\Delta t = rnd(T_{Mess}/i) - T_{Mess}/i$. Aufgrund der Vorsage $T_{Vor} = (m - 1)\cdot T_{Slot, M}$, bewirkt er einen Gesamtfehler $\Delta T^*_{Vor} = (m - 1)\cdot[rnd(T_{Mess}/i) - T_{Mess}/i]$. Dieser liegt bei großem m deutlich über dem Gesamtfehler $\Delta T_{Vor} = rnd[(m - 1)\cdot T_{Mess}/i] - (m - 1)\cdot T_{Mess}/i$ mit Formel 4.9. Ein direkter Vergleich zeigt: $\Delta T^*_{Vor} - \Delta T_{Vor} = (m - 1)\cdot rnd(T_{Mess}/i) - rnd[(m - 1)\cdot T_{Mess}/i]$. Daraus lässt sich der max. mögliche Unterschied für $m >> 1$ abschätzen: $|\Delta T^*_{Vor} - \Delta T_{Vor}|_{max} \approx (m - 2)\cdot \Delta T_{Vor, max}$. Es geht klar her-

[92] Sie zeigt unter anderem, dass die Vermessung eines alternierenden Bitmusters in einem „Frame“, wie z.B. beim LIN-Bus, schlichtweg zu kurzfristig und damit für „Basic-TTP/A“ ungeeignet ist.

vor, dass die einfachere Vorhersage mit $T_{Slot, M}$ nur wenige „Slots“ ausreichend genau ist, weshalb bei längeren Runden regelmäßige Korrekturen erforderlich sind.

Die „Slot“-Vorhersage nach Formel 4.9 kennt dieses Problem nicht. Dafür benötigt sie mehr Ressourcen, wenn ein „Slave“ viele „Slots“ in einer Runde belegt. Hauptsächlich schlagen die erforderliche „Timer“-Breite, die Arithmetik und der Speicher für die Vorhersagewerte zu Buche – eine Berechnung bei Bedarf ist i.A. zu langsam. Wenn ein „Slave“ den letzten „Slots“ in einer max. langen Runde belegt, benötigt er bei der Minimalauflösung $\Delta T_S = T_{Bit, M}/8$ einen „Timer“ mit $253 \cdot 13 \cdot 8 = 26312$ Quantisierungsschritten zum Vorhersagen – für die Messung sind nur unbedeutend weniger Quantisierungsschritte notwendig. Da einige „low-cost“ µCs nur einen 8 Bit breiten „Timer“ besitzen, muss man mit einer Erweiterung per Software rechnen. Außerdem ist mindestens eine 24 Bit-Arithmetik erforderlich, damit T_{Vor} mit nur einer Ganzzahldivision berechnet werden kann. Die Forderung entsteht aufgrund des Zählers in Formel 4.9. Er umfasst im Wesentlichen das Produkt des 8 Bit-Werts (m - 1) und des 16 Bit-Werts T_{Mess}. Bezüglich der Programmierung stellt das kein Problem dar, weil jede vernünftige C-Bibliothek eine erweiterte Arithmetik unterstützt. Angesichts der Messperiode von 2..3 s, gilt das Gleiche für die Rechenzeit. Beim Speicherbedarf wird es schwieriger. „Low-cost“ µCs besitzen oftmals nur wenig Speicher. Deshalb besteht die Möglichkeit, dass die Gangfehlerkorrektur, bei vielen „Slots“ in einer Runde, den einen oder anderen Baustein überfordert. In diesen Fällen muss entweder etwas mehr für den nächst größeren „low-cost“ µC oder einen Quarzoszillator bezahlt werden. Alternativ kann man die Gangfehlerkorrektur abspecken und einen günstigeren Keramikoszillator verwenden oder die max. Rundenlänge reduzieren.

4.1.4 UART-Synchronisation

Am Anfang von Abschnitt 4.1.3 ist beschrieben, dass „Slaves“ mit RC- oder „On-Chip“-Oszillatoren einen Frequenzfehler von ca. ±(5..10) % aufweisen können. Der Frequenzfehler wirkt sich direkt auf die Baudrate des UARTs aus und überschreitet die zulässigen Toleranzgrenzen von ±1,84 % nach Abschnitt 4.1.1. Deshalb müssen diese „Slaves“, neben dem „Slot-Timer“-Gangfehler, auch die Baudrate korrigieren – auf den Lagefehler des UARTs haben sie i.A. keinen Einfluss (siehe ebenfalls Abschnitt 4.1.1). Bei „Slaves“ mit Keramikoszillatoren, deren Frequenzfehler betragsmäßig unter 1 % liegt, ist das i.A. nicht nötig. Sie können, wie „Slaves“ mit Quarzoszillatoren, mit einer A-Priori-Baudrate arbeiten.

Die Synchronisation des Bittakts baut auf die Messungen der „Slot-Timer“-Gangfehlerkorrektur und die Gesetzmäßigkeit $T_{Bit} = T_{Slot}/13$. Entsprechend den Ausführungen in Abschnitt 4.1.3, kann ein „Slave“ die „Slot“-Dauer des „Masters“ über $T_{Slot, M} \approx T_{Mess}/i$ abschätzen. Folglich kennt er auch in etwa den Soll-Bittakt $T_{Bit, M} \approx T_{Mess}/(i \cdot 13)$. Aufgrund der Granularität des UART-„Timers“ ΔT_{UART}, wird ein „Slave“ den Wert $T_{Mess}/(i \cdot 13)$ nur selten exakt einstellen können. Deshalb muss ein Einstellfehler ΔT_{Quant} mit der Spannweite ΔT_{UART} berücksichtigt werden. Mit einer Rundungsoperation gemäß Formel 4.12, lässt er sich gleichmäßig um den Nominalwert $T_{Mess}/(i \cdot 13)$ verteilen.

$$T_{Bit, S} = \mathrm{rnd}\left(\frac{T_{Mess}}{i \cdot 13 \cdot \Delta T_{UART}} \right) \cdot \Delta T_{UART}$$

Formel 4.12: UART-Synchronisation in einem „Slave“ mit Gangfehlerkorrektur

In den Restfehler fließt also der Einstellfehler ΔT_{Quant} und der Messfehler $\Delta T_{Mess, ges}$ ein. Letzterer ist aus Abschnitt 4.1.3 ebenso bekannt wie die Herleitung des Restfehlers. Deswegen entfallen hierzu die

Erläuterung und es wird unmittelbar das, in Formel 4.13 zu sehende, Ergebnis präsentiert. Eine erste Konsequenz entsteht aus den Tatsachen $\Delta T_{UART} > 0$ und $T_{Bit, M} > 0$: Damit die Ungleichung überhaupt erfüllbar ist, muss $i > 1{,}046..$ sein. Folglich ist die Untergrenze für die Dauer einer Messung $2 \cdot T_{Slot, M}$. Auswirkungen für den Betrieb hat die Untergrenze keine, weshalb ihre Bedeutung rein theoretischer Natur ist. Denn erstens kann, aufgrund der IRG, eine Messung nicht kürzer dauern und zweitens ist eine so kurze Messung nicht praxisrelevant. Typische Werte für i werden zwischen zehn und 100 liegen. Dadurch verliert der Messfehleranteil, gegenüber dem Anteil des Einstellfehlers, stark an Bedeutung. Vereinfachend gilt demnach: $\Delta T_{UART, max} \approx 30 \cdot T_{Bit, M}$. Die Auflösung eines UARTs ΔT_{UART} wiederum ist indirekt proportional mit der Oszillatorfrequenz verknüpft. Hieraus ergibt sich die zweite Konsequenz: Falls ein „Slave“ eine UART-Synchronisation benötigt, muss seine Oszillatorfrequenz sehr viel größer sein als die gewünschte Baudrate.

$$\left|\Delta T_{Bit, S}\right|_{max} \approx \frac{\Delta T_{Mess, ges, max}}{i \cdot 13} + \Delta T_{Quant, max} \leq 0{,}0184 \cdot T_{Bit, M}$$

$$\Delta T_{Mess, ges, max} \approx \frac{4}{16} \cdot T_{Bit, M} \qquad \Delta T_{Quant, max} = \frac{\Delta T_{UART}}{2}$$

$$\Rightarrow \qquad \frac{\Delta T_{UART}}{T_{Bit, M}} \leq 0{,}0368 - 0{,}0385 \cdot \frac{1}{i}$$

Formel 4.13: Restfehler der UART-Synchronisation

Die Überprüfung der Baudrate findet am besten mit der Gangfehlerkorrektur des „Slot-Timers“ statt (alle 2..3 s). Eine Aktualisierung ist jedoch weitaus seltener erforderlich, weil die Granularität und das Toleranzband der Baudrate größer ist als bei der „Slot“-Periode[93]. Allerdings darf die UART-Synchronisation, im Gegensatz zur „Slot-Timer“-Gangfehlerkorrektur, nur in kommunikationsfreien Phasen (z.B. IRG) durchgeführt werden. Während einer Kommunikation besteht die Gefahr einer nachhaltigen Störung des Sende- und/oder Empfangszykluses. Weitere Details hierzu befinden sich in Abschnitt 4.2 und im Speziellen in Abschnitt 4.2.6.

4.1.5 „Boot“-Vorgang

Dieser Abschnitt betrifft ausschließlich „Slaves“ mit RC-, „On-Chip“- oder ähnlich ungenauen Oszillatoren. Aus dem Vorangehenden ist bekannt, dass solche „Slaves“ durchaus einen Frequenzfehler von ±(5..10) % haben können und eine UART-Synchronisation benötigen, damit ihr Baudratenfehler innerhalb des Toleranzbandes mit den Grenzen ±1,84 % bleibt. Bedingung für die UART-Synchronisation ist eine funktionierende „Slot-Timer“-Gangfehlerkorrektur. Die fordert ihrerseits einen fehlerfreien Empfang der UART-„Frames“, woraus ein „Boot“-Problem resultiert. Nach einem „Reset“ schlägt der volle Frequenzfehler auf die Baudrate durch. Fällt er zufälligerweise groß aus, empfängt ein „Slave“ nur falsche „Frames“. Folglich kann er den Abstand zwischen den „Fireworks-Frames“ nicht messen und deshalb weder seine „Slot-Timer“-Gangfehlerkorrektur, noch seine UART-Synchronisation starten. Der „Slave“ befindet sich in einer Pattsituation.

[93] Aus diesem Grund ist das Verhältnis T_{Slot}/T_{Bit} in einem „Slave“ mit ungenauem Oszillator (RC, „On-Chip“, Keramik) i.A. ungleich 13, wodurch sich die unter Punkt 3.2.4 erwähnte lose Kopplung erklärt.

Um dort herauszufinden, benötigt er eine automatische Baudratenjustierung. Das Prinzip ist in Bild 4.7 dargestellt. Es funktioniert so ähnlich wie die weitverbreitete Baudratendetektion bei Modems, ist jedoch einfacher und schneller. Zwei Modems handeln stets die höchste Baudrate aus. Dabei spielen die Verbindungsqualität und die unterstützten Geschwindigkeiten eine Rolle. Aufgrund der vielfältigen Möglichkeiten, kann dieser Vorgang viele Sekunden dauern. Bei „Basic-TTP/A“ hingegen wird die Baudrate beim Entwurf einer Applikation festgelegt. Wenn man z.B. einen Regelkreis über den Bus schließt, bestimmen die Abtastzeit und Rundenlängen die Baudrate (siehe Abschnitt 5.4). Folglich darf ein „Slave“ keine Baudrate aushandeln, sondern er muss die geplante Baudrate schlichtweg einstellen. Da das für alle „Slaves“ gilt, besteht bis zur Box „set application baudrate“ in Bild 4.7 kein Unterschied zwischen einem „Slave“ mit Quarz- oder Keramikoszillator und einem „Slave“ mit RC-, „On-Chip“- o.Ä. Oszillator.

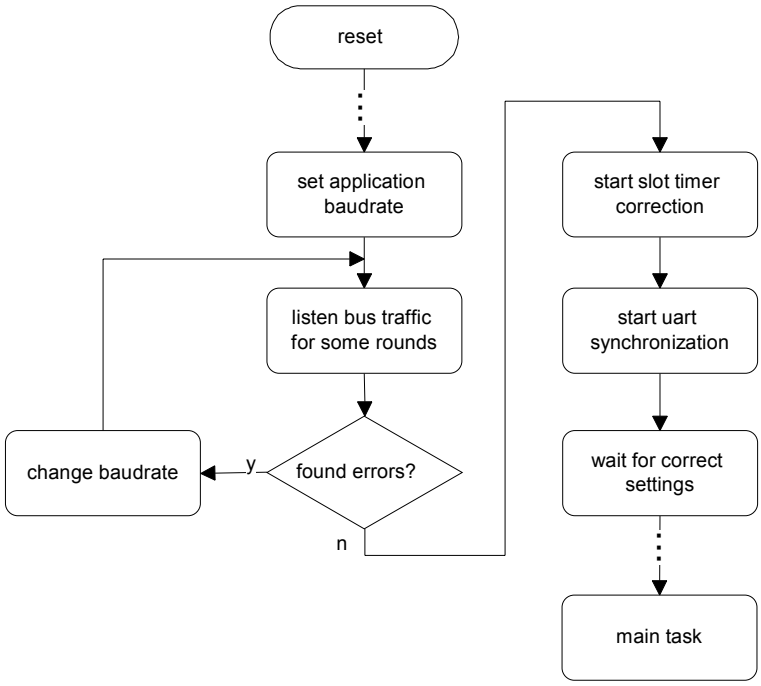


Bild 4.7: Prinzip der Baudratenjustierung bei „Slaves“ mit ungenauen Oszillatoren

Letzterer kann sich allerdings nicht darauf verlassen, dass seine Baudrate ausreichend genau ist. Aus diesem Grund hört er am Bus vorerst nur mit und ändert seine Baudrate so oft, bis er einige Runden lang keine Fehler entdeckt. Weil sich die Startbaudrate bereits in der Nähe der Sollbaudrate befindet, ist der Suchvorgang in „Basic-TTP/A“ einfach und schnell – Algorithmen hierfür lassen sich aus den zahlreichen Schriften zum Thema automatische Baudratenerkennung ableiten (z.B. [146], [139], [134]). Danach aktiviert der „Slave“ seine „Slot-Timer“-Gangfehlerkorrektur und UART-Synchronisation. Bevor er jedoch am Busgeschehen mit der geforderten Präzision teilnehmen kann, muss er den ersten Messung- und Korrekturzyklus verstreichen lassen. Ab diesem Zeitpunkt halten sich die „Slot-Timer“-Gangfehlerkorrektur und UART-Synchronisation gegenseitig am Leben, sofern beim Design des „Slaves“ die Punkte 4.1.3 und 4.1.4 beachtet wurden. Die „Slot-Timer“-Gangfehlerkorrektur informiert die UART-Synchronisation über den Frequenzfehler, worauf Letztere die Baudrate nachführt und Erstere weitere Messungen ermöglicht.

4.1.6 Netzausdehnung

Prinzipiell gibt es bei der Ausdehnung eines TTP/A-Netzes zwei begrenzende Faktoren: Einmal die Signalabschwächung und das andere Mal die Signallaufzeit. Die Signalabschwächung hängt von der Länge und Dämpfung des Kabels und von der Anzahl der Knoten ab. Sie hat bei TTP/A keine große Bedeutung, weil es sich um einen Sensor/Aktorbus handelt. Laut Kapitel 2, beträgt deren Kabellänge 10..100 m und die max. Knotenanzahl 30..40 – bei diesen typischen Daten, benötigt man in der Regel noch keinen „Repeater“ für die Signalregeneration. Deshalb ist für TTP/A die Signallaufzeit in Sachen Netzausdehnung maßgebend. Wegen der hohen Ausbreitungsgeschwindigkeit elektrischer Signale, spielt sie erst bei höheren Baudraten eine Rolle. Bei TTP/A ist das ab ungefähr 200..300 kBit/s der Fall, wie die folgenden Ausführungen zeigen werden.

Aus den Abschnitten 4.1.1 und 4.1.2 ist bekannt, dass eine automatische Korrektur der Signallaufzeit die Möglichkeiten vieler „low-cost“ µCs übersteigt und deshalb ein Zeitraum von $0{,}16 \cdot T_{Bit,\,M}$ in der IFG für ihre Kompensation reserviert wurde. Es ist jedoch unklar, welchen Anteil die Leitungslaufzeit darin belegen darf. Folglich muss zuerst ein entsprechendes „Worst-Case“-Szenario untersucht werden. Hierzu ist das Busgeschehen in Bild 4.8, bei einem örtlich getrennten „Master“ und „Slave“, dargestellt. Wie üblich, sendet der „Master“ zu Beginn einer Runde ein „Fireworks-Frame“. Dieser gelangt, aufgrund der notwendigen Busanschaltung, um $T_{D,\,BA}$ verzögert auf den Bus. Anschließend breitet er sich auf der Leitung aus und erreicht nach der Leitungslaufzeit $T_{D,\,Ltg}$ den Ort des „Slaves“. Da der „Slave“ ebenfalls eine Busanschaltung besitzt, vergehen weitere $T_{D,\,BA}$, bis er den „Fireworks-Frame“ registriert – es werden gleiche Verzögerungen $T_{D,\,BA}$ angenommen und keine anderen Zeitfehler berücksichtigt. Somit beträgt der signallaufzeitbedingte Lagefehler des „Slot-Timers“ in dem „Slave“ $T_{D,\,Sig} = T_{D,\,Ltg} + 2 \cdot T_{D,\,BA}$.

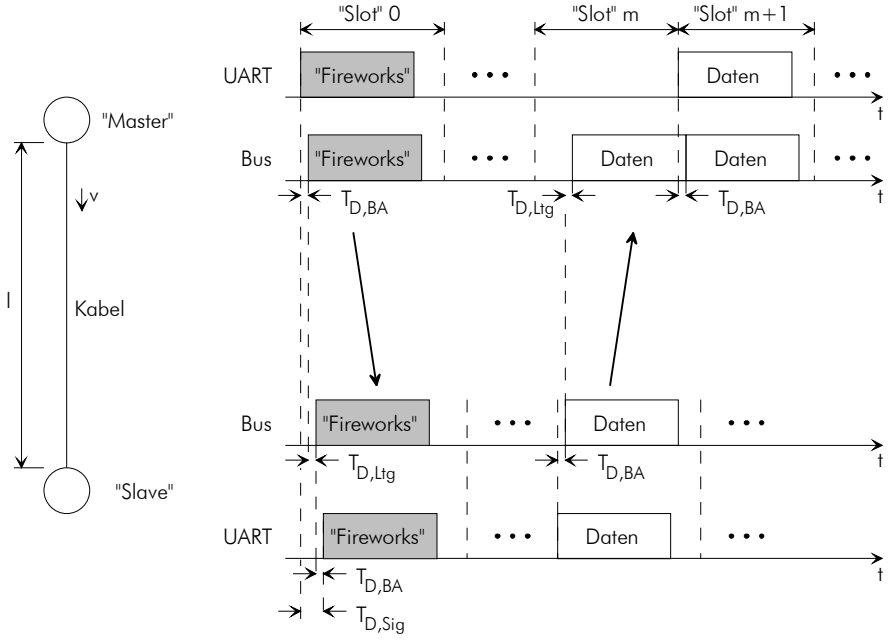


Bild 4.8: Maximale Signallaufzeit in „TTP/A“

Mit dieser Verzögerung sendet der „Slave“ in „Slot“ m ein Daten-„Frame“. Bis dieser am Ort des „Masters“ ankommt, dauert es eine weitere Zeitspanne von $T_{D,\,Ltg} + T_{D,\,BA}$. Wenn der „Master“ im nächsten „Slot“ schreibt, tritt der „Worst-Case“ ein. Es besteht die Gefahr einer Buskollision zwischen

den Daten-„Frames“ in „Slot“ m und m + 1 am Ort des „Masters“. Schließlich verspäten sich die Daten-„Frames“ des „Slave“ hier am meisten. Man muss jedoch nicht die volle Verspätung $2 \cdot T_{D, Ltg} + 3 \cdot T_{D, BA}$ beachten, da auch die „Frames“ des „Masters“ durch seine Busanschaltung um $T_{D, BA}$ verzögert am Konfliktort erscheinen.

Folglich ergibt sich, unter der Annahme voll ausgereizter Toleranzen, als Restriktion für die Signallaufzeit: $0{,}16 \cdot T_{Bit, M} \geq 2 \cdot T_{D, Ltg} + 2 \cdot T_{D, BA}$. Die Verzögerung $T_{D, BA}$ hängt vom verwendeten „Physical-Layer“ (RS485, CAN, ISO9141) und den „Transceiver“-Bausteinen (MAX485, DS96176, TLE6250...) ab. Ihre Varianz ist jedoch nicht so groß, als dass man sie nicht abschätzen kann. In erster Näherung lässt sich für $T_{D, BA}$ ein Wert von 50 ns angeben[94]. Des Weiteren gilt für die Leitungslaufzeit $T_{D, Ltg} = l/v$, mit $v \approx 2{,}2 \cdot 10^8$ m/s (typische Ausbreitungsgeschwindigkeit elektrischer Signale auf Kupferkabel[95]). Durch Einsetzen dieser Zwischenergebnisse in die obige Restriktion und Auflösen nach der Leitungslänge erhält man: $l_{max} \approx 2{,}2 \cdot 10^8$ m/s$\cdot(0{,}08 \cdot T_{Bit, M} - 100$ ns). Eine aussagekräftigere graphische Auswertung der Formel zeigt Bild 4.9.

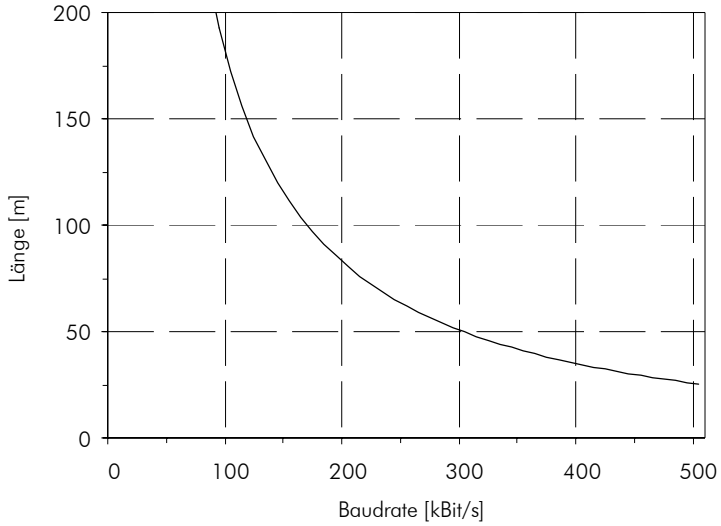


Bild 4.9: Maximale Netzausdehnung in Abhängigkeit der Baudrate

Wie anfangs erwähnt, muss der Netzausdehnung bei TTP/A erst ab 200..300 kBit/s Beachtung geschenkt werden. Da jedoch nur wenige Sensor/Aktorbusanwendungen eine höhere Baudrate benötigen, existiert diesbezüglich keine praktisch Einschränkung. Höhere Baudraten werden voraussichtlich nur in der Kfz-Technik benötigt. In dieser Sparte liegen die Sensor/Aktorbusausdehungen allerdings nur im unteren 10 m-Bereich, so dass ein Betrieb von TTP/A bis 500 kBit/s und mehr durchaus Sinn macht.

---

[94] Eine galvanische Trennung (Optokoppler o.Ä.) ist in der Regel nicht erforderlich, da die Netzausdehnung eines Sensor/Aktorbusses klein ist. Außerdem würde eine galvanische Trennung die Kosten pro Knoten erheblich steigern.

[95] $v = \omega/\beta = (L' \cdot C')^{-0,5} = 1/(Z_0 \cdot C')$, mit $Z_0 \approx 75\ \Omega$ und $C' \approx 60$ pF/m.

4.2 Programmkonzept

4.2.1 Problematik

„Basic-TTP/A“ wirkt auf den ersten Blick einfach. Das verleitet zur Unterschätzung der Probleme bei einer Realisierung. Hierzu gehören die knappen Ressourcen von „low-cost“ µCs, die großen Toleranzen günstiger Oszillatoren, die zahlreichen Schmutzeffekte (UART-„Jitter“, „Critical-Sections“, Startbiterkennung, zeitvariante Befehle...) und die hohen Realzeitanforderungen. Des Weiteren spielt die enge Verzahnung der „Basic-TTP/A“-Mechanismen eine Rolle. Sie gestattet leider kein Zusammensetzen von isolierten Teillösungen zu einer Gesamtlösung. Damit erreicht man zwar Teilerfolge, verfehlt letztendlich aber das Ziel. Stattdessen müssen sämtliche Probleme in einem Schritt gelöst werden. Alles zusammen macht eine Abbildung von „Basic-TTP/A“ in Software komplex. Das haben die Erfahrungen in dieser Arbeit gezeigt und lassen die vorangehenden Ausführungen, insbesondere die über die Zeitfehler, erahnen.

Glücklicherweise ist es gelungen ein entsprechendes Programmkonzept zu entwickeln. Es ermöglicht eine effiziente, ressourcenschonende Implementierung von „Basic-TTP/A“ und verletzt auch bei hohen Geschwindigkeiten keine „Deadlines“. Der Schlüssel ist ein so genannter „Interleave“-Algorithmus mit Selbstjustierung. Er nimmt auf die Schmutzeffekte Rücksicht und vermeidet „Overhead“. Außerdem spielt die „Task“-Aufteilung für die optimale Nutzung der Rechenzeit und des Speichers eine Rolle. Die Leistungsfähigkeit dieses Programmkonzepts wurde mit einer Referenzimplementierung in C auf C16x-µCs unter Beweis gestellt. Ohne Codeoptimierung ist ein Betrieb bis 625 kBit/s (Grenze des µCs) möglich. Zum Vergleich: In den sonstigen Versuchen lagen die höchsten Baudraten, trotz rudimentärer Kommunikation, deutlich unter 100 kBit/s. Hinzu kommt dass, dieses Konzept die Nebenläufigkeit der Applikation unterstützt, unproblematisch in der Handhabung ist und sich in einigen Anwendungsbeispielen (siehe Kapitel 5) bewährt hat. Deshalb verwenden es die Projektpartner SiemensVDO und Universität Stuttgart als Basis für ihre Forschungsarbeiten weiter.

4.2.2 „Task“-Aufteilung

Ereignisorientierter Ansatz

Die Abbildung in Software macht „Basic-TTP/A“ zu einem reaktiven System. Es muss auf bestimmte Ereignisse des µCs o.Ä. in einer gewissen Zeit angemessen reagieren. Zu diesen Ereignissen zählen der „Slot-Timer-Interrupt“ und die „Interrupts“ des UARTs (Hardware oder Emulation). Jeder UART besitzt einen „Receive-Interrupt“ und viele zusätzlich einen „Transmit-“ und „Error-Interrupt“. Der „Receive-Interrupt“ zeigt den Empfang eines „Frames“ an. Er wird i.A. ungefähr in der Mitte des Stoppbits ausgelöst. Mit dem „Transmit-Interrupt“ signalisiert ein UART, dass ein „Frame“ gesendet wurde. Sofern dieser Fall zutrifft, erscheint der „Transmit-Interrupt“ am Anfang des Stoppbits. Das hat folgende Gründe: Zum einen muss das Stoppbit nicht aktiv gesendet werden und zum anderen gewinnt man Zeit zum Nachladen für einen lückenlosen Sendebetrieb. Und schließlich der „Error-Interrupt“. Hierbei handelt es sich meistens um einen Sammel-„Interrupt“, der bei einem „Frame-“, „Parity-“ oder „Overrun-Error“ ausgelöst wird. Der „Error-Interrupt“ tritt entweder mit oder kurz nach dem „Receive-Interrupt“ auf[96].

[96] Im „low-cost“ Segment gibt es ein paar µCs mit UARTs, die nur einen „Receive-Interrupt“ besitzen. Hier müssen die „Error-Flags“ des UARTs explizit abgefragt werden. Falls man einen „Transmit-Interrupt“ benötigt, ist das Mithören der eigenen „Frames“ und eine indirekte Reaktion im „Receive-Interrupt“ erforderlich. Für „Basic-TTP/A“ entstehen daraus keine Nachteile.

Es liegt nahe, auf jeden dieser vier „Interrupts“ mit einer entsprechenden „Tasks“ unmittelbar zu reagieren – hier wird nicht zwischen einer ISR, einer „Task“, einem „Thread“ oder einem Prozess unterschieden. Zumindest war das bei den erwähnten, unabhängig durchgeführten Versuchen der Fall. Man erreicht mit diesem ereignisorientierten Ansatz minimal kurze Reaktionszeiten, was für ein Realzeitsystem, in der Regel zuträglich ist. Die Aufgaben dieser „Tasks“ sind in Bild 4.10 grob skizziert. Sie werden im Folgenden beschrieben, ohne auf Details oder exakte Zusammenhänge einzugehen. Dafür ist die Anhängigkeit vom Knotentyp („Master“ oder „Slave“) und der Implementierungsart (Mithören aller „Slots“ oder Zeitfensterung) zu groß. Außerdem ist das Ziel eine Abschätzung der Rechenzeiten zum Zweck einer Bewertung.

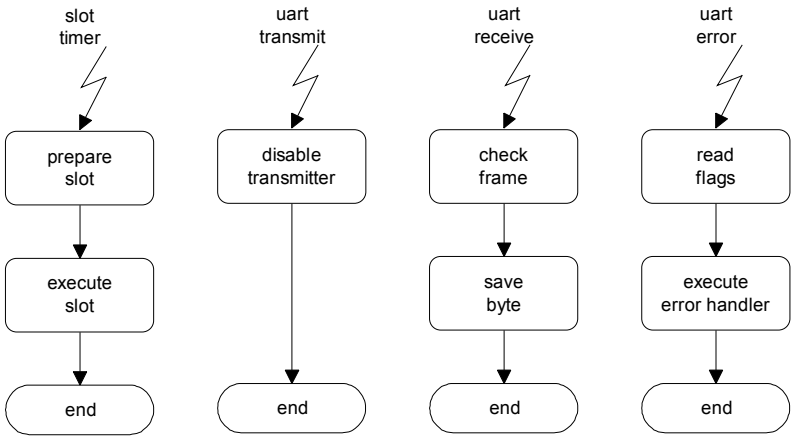


Bild 4.10: Ereignisorientierte „Task“-Aufteilung

In der „Slot-Task“ muss der Knoten als erstes im „Schedules“ nachsehen, damit er weiß, welche Aufgaben anstehen. Wenn es sich um einen Schreib-„Slot“ handelt, kopiert die „Slot-Task“ das entsprechende Datenbyte in den UART und aktiviert den „Transmitter“. Bei einem Lese-„Slot“ kann sie den „Receiver“ aktivieren und der „Receive-Task“ mitteilen, wo das empfangene Byte ablegt werden soll. Des Weiteren prüft die „Slot-Task“, ob im letzten „Slot“ ein „Frame“ empfangen wurde, sofern es ein Lese-„Slot“ war. Sollte der erwartete „Frame“ fehlen, informiert die „Slot-Task“ die „Error-Task“ über diesen Fehler. Anschließend terminiert die „Slot-Task“. Die „Transmit-Task“ wird bei einem „Tristate-Physical-Layer“ (z.B. RS485) nach dem Senden eines „Frames“ aktiv. Sie schalten den „Transmitter“ in den hochohmigen Zustand und wartet auf den nächsten „Transmit-Interrupt“.

Aufwändiger ist dagegen wieder die „Receive-Task“. Für eine zusätzliche Fehlersicherung führt sie in einem Schreib-„Slot“ einen Vergleich zwischen dem gesendeten und dem empfangenen Byte durch. Bei Ungleichheit benachrichtigt sie die „Error-Task“. Handelt es sich um einen Lese-„Slot“, so wird das empfangene Byte, nach einer erfolgreich durchgeführten Parity-Prüfung, gespeichert. Eine erfolglose Parity-Prüfung führt zur Verwerfung des empfangenen Bytes und zu einer entsprechenden Meldung an die „Error-Task“. Bevor die „Receive-Task“ terminiert, können z.B. noch Synchronisationsaktionen mit der Applikation durchgeführt werden. Für das Sammeln der Fehler ist die „Error-Task“ zuständig. Wie angedeutet, können in „Basic-TTP/A“ nicht nur UART-Fehler, sondern auch Vergleichs- und Ablauffehler eintreten. Die Auswertung der Fehler und die Reaktionen darauf finden größtenteils in einer applikationsspezifischen Fehlerbehandlungsroutine statt.

Aus dieser rudimentären Beschreibung kann man bereits Rückschlüsse auf die Rechenzeiten ziehen. Die „Slot“- und „Receive-Task“ benötigen am meisten, woraus eine qualitative Belegung der CPU

gemäß Bild 4.11 entsteht. Es ist leicht zu erkennen, dass die ereignisorientierte „Task“-Aufteilung in „Basic-TTP/A“ ein Konfliktpotenzial beherbergt. Am Ende eines „Slots“ steht wenig Rechenzeit zur Verfügung und es können viele Ereignisse auftreten. Da die benötigte Rechenzeit konstant ist und der Spielraum mit steigender Baudrate kleiner wird, kollidiert die „Receive-Task“ früher oder später mit der nächsten „Slot-Task“. Prinzipiell besteht diese Gefahr auch bei der „Error-Task“. Sie bleibt jedoch außen vor, weil das erstens nur im Fehlerfall relevant ist und zweitens von der applikationsspezifischen Fehlerbehandlungsroutine abhängt.

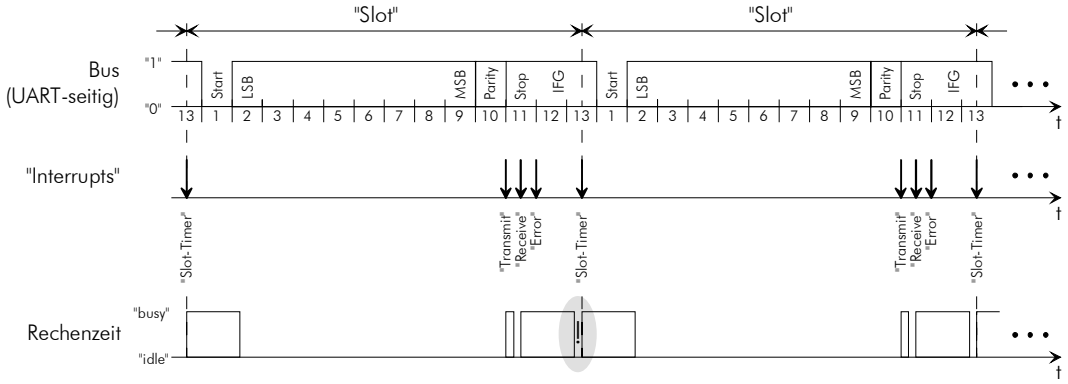


Bild 4.11: Rechenzeiten bei einer ereignisorientierten „Task“-Aufteilung

Bei einer Kollision zwischen der „Receive-“ und „Slot-Task“ muss ein „Task“-Wechsel stattfinden. Eine Verzögerung der „Slot-Task“ ist nicht zulässig, da im Sendefall die Realzeitbedingungen verletzt werden können. Die Unterbrechung der „Receive-Task“ ist jedoch problematisch. Schließlich benutzen beide „Tasks“ dieselben Betriebsmittel, Dadurch ist mit Ressourcenkonflikten und Dateninkonsistenzen zu rechnen, falls keine zusätzlichen Maßnahmen getroffen werden. Unter diesen Umständen kann z.B. der Sende-Empfangs-Vergleich bei zwei aufeinanderfolgenden Sende-„Slots“ (z.B. 16 Bit-Wert) gestört werden. Es besteht die Möglichkeit, dass eine Unterbrechung an entsprechender Stelle fälschlicherweise zu einem Vergleich zwischen dem empfangenen Byte aus dem vorderen „Slot“ und dem in Sendung befindlichen Byte aus hinteren „Slot“ führt.

In einem Lese-„Slot“ dagegen kann eine Unterbrechung der „Receive-Task“ eine Speicherung des empfangenen Bytes an einer falschen Stelle bewirken. Dieser Fall tritt ein, wenn die „Receive-Task“ den Speicherort anhand des aktuellen „Slots“ ermittelt und der entsprechende Vorgang im nächsten „Slot“ ausgeführt wird. Falls die „Receive-Task“ diese Information von der „Slot-Task“ erhält, besteht diese Möglichkeit ebenso. Dazu müssen lediglich zwei Lese-„Slots“ aufeinanderfolgen, der Speichervorgang des vorderen „Slots“ im anschließenden stattfinden und die wiederaufgenommene „Receive-Task“ die überschriebene Speicherortinformationen lesen. Ähnliches ist auch bei der „Inter-Task“-Kommunikation (z.B. mit der „Error-Task“), während eines Synchronisationsvorgangs mit der Applikation und vielem mehr denkbar.

Die Ursachen dieser Fehler sind immer die Gleichen. Es handelt sich um die erwähnte Nutzung gemeinsamer Ressourcen durch konkurrierende „Tasks“. Da dieses Thema von den „Multitasking“-Betriebssystemen her bekannt ist, bestünde die Möglichkeit auf eine der vorhandenen Lösungsstrategien zurückzugreifen. Allerdings scheitert dieses Vorhaben an den knappen Ressourcen der „low-cost“ µCs. Ein Stück weiter oben wurde erwähnt, dass die „Slot-Task“ nicht verzögert werden darf. Deshalb entfallen alle Lösungsstrategien, die auf dem Prinzip der Sperrsynchronisation beruhen (Semaphor, „Mutual-Exclusion“...). Somit bleiben die kooperativen Synchronisationsverfahren („Mailbox“, „Mes-

sage Queue“...) übrig. Da diese Art der Synchronisation mit Datenpuffern (FIFO o.Ä.) arbeitet und einen hohen Verwaltungsaufwand produziert, eignet sie sich nur für Systeme mit reichlich Speicher und viel Rechenleistung. Folglich dürfen die „Receive-“ und „Slot-Task“ nicht kollidieren.

Das Ganze wäre alles halb so schlimm, läge die max. Baudrate vernünftig hoch. Die Limitierung tritt jedoch weit unter 100 kBit/s ein. Anders als in Bild 4.11 dargestellt, kann der „Receive-Interrupt“ sehr dicht an das „Slot“-Ende heranrücken. Den Ausführungen unter Punkt 4.1 nach, darf er sogar unmittelbar vor dem „Slot-Timer-Interrupt“ des nächsten „Slots“ liegen. In diesem äußerst pessimistischen Fall, bei dem alle Toleranzen voll ausgereizt werden, würde die max. Baudrate Null betragen. Da dieses Szenario eher unwahrscheinlich ist, vor allem bei niedrigen Baudraten, wird der Spielraum zwischen „Receive-“ und „Slot-Timer-Interrupt“ mit ca. ¼..½·$T_{Bit, M}$ realistischer abgeschätzt – Knoten mit RC- oder „On-Chip“-Oszillatoren tendieren mehr zur Untergrenze, Knoten mit Keramik- oder Quarzoszillatoren können auch einen größeren Spielraum haben. Für eine Abschätzung der Rechenzeit wird die C16x-Implementierung herangezogen. Als Grundlage dienen die Oszillogramme in Bild 4.13 (siehe S. 105). Sie zeigen die Rechenzeiten einer „Task“-Aufteilung, bei der die Aufgaben der „Slot-“ und „Receive-Task“ in einer Haupt-„Task“ zusammengefasst sind; weitere Erläuterung im Anschluss. Da die Aufgaben der „Slot-“ und „Receive-Task“ in erster Näherung gleiches Gewicht haben, kann die Rechenzeit der „Receive-Task“, zuzüglich 1 µs für die fehlenden Wechselzeiten, mit (½·10 +1) µs = 6 µs abgeschätzt werden[97]. Somit ergibt sich bei einer ereignisorientierten „Task“-Aufteilung mit einem C16x-µC eine max. Baudrate von ungefähr 40..80 kBit/s.

Viele „low-cost“ µCs haben allerdings weniger Rechenleistung als der 10 MIPS starke C16x-µC. Deren durchschnittliche Rechenleistung liegt momentan eher bei 5 MIPS. Beispiele hierfür sind die „low-cost“ µCs von Microchip, Philips, Zilog, ST und Cypress. Lediglich Atmel hat ein paar „low-cost“ µCs, die mit der Rechenleistung des C16x-µCs gleich auf ziehen[98]. Deshalb kann die max. Baudrate auch deutlich unter 40 kBit/s liegen.

Rechenzeitorientierter Ansatz

Wesentlich bessere Ergebnisse erzielt man mit einer rechenzeitorientierten „Task“-Aufteilung. Schließlich ist die Auslastung der CPU nicht das Problem (siehe Bild 4.11). Sie beträgt bei der ereignisorientierten „Task“-Aufteilung in einem aktiven „Slot“ ungefähr 5..10 %. Damit liegt die Idee des rechenzeitorientierten Ansatzes auf der Hand. Wenn es gelingt, die Aufgaben geschickter zu verteilen, ihre Bearbeitung also soweit als möglich in den großen Freiräumen stattfindet, dann treten die „Task“-Konflikte erst bei viel höheren Baudraten auf. Es kommt zugute, dass „Basic-TTP/A“ einen TDMA-Anteil besitzt, wodurch das Geschehen in den Runden von vornherein bekannt ist. Aus diesem Grund korrelieren die Ereignisse. Nach einem „Slot-Timer-Interrupt“ folgt ein „Transmit-Interrupt“, sofern gesendet wurde, in jedem Fall folgt aber ein „Receive-Interrupt“ und im Fehlerfall ein „Error-Interrupt“. Solange ein Knoten an den „Slots“ teilnimmt, wiederholt sich dieser Ablauf. Von dieser Situation soll vorerst ausgegangen werden.

Sie gestattet einem Knoten nicht unmittelbar auf alle UART-Ereignisse am Ende eines „Slots“ reagieren zu müssen. Kurz danach folgt ja der nächste „Slot-Timer-Interrupt“. Mit entsprechenden Algorithmen eignet er sich zum Reagieren auf zeitunkritische UART-Ereignisse ebenso. Dazu zählen der „Receive-“ und „Error-Interrupt“, weil ein empfangenes Byte oder ein aufgetretener Fehler bis zum Ende des nächsten „Slots“ nicht verloren gehen. Für die Applikationen spielt es in der Regel auch

---

[97] In dieser Zeit bearbeitet der C16x in etwa 60 Maschinenbefehle (≈ 15..20 C-Befehle). Das ist für die Aufgaben der „Receive-Task“ durchaus relevant.

[98] Von Dallas gibt es 8051-Derivate (f_{CPU} ≈ 100 MHz) mit wesentlich mehr Rechenleistung. Diese superschnellen, kleinen Bausteine kosten allerdings auch mehr, als die „low-cost“ µC von Atmel, Microchip, Philips etc.

keine Rolle, ob sie ein Datum oder eine Fehlermeldung unmittelbar oder leicht verzögert erhalten. Nicht verschiebbar ist hingegen die „Transmit-Task“. Sie muss beim Auftreten des „Transmit-Interrupts“ sofort aktiviert werden, falls ein „Tristate-Physical-Layer“ (z.B. RS485) verwendet wird. Ansonsten ist der Bus nach einem Sende-„Slot“ nicht rechtzeitig frei. Die Rechenzeit der „Transmit-Task“ ist jedoch sehr kurz, weil ihr Kontext und ihre Aufgabe minimal sind. Folglich können bei einer rechenzeitorientierten „Task“-Aufteilung die „Slot-“, „Receive-“ und „Error-Task“, so wie in Bild 4.12 skizziert, zusammengefasst werden.

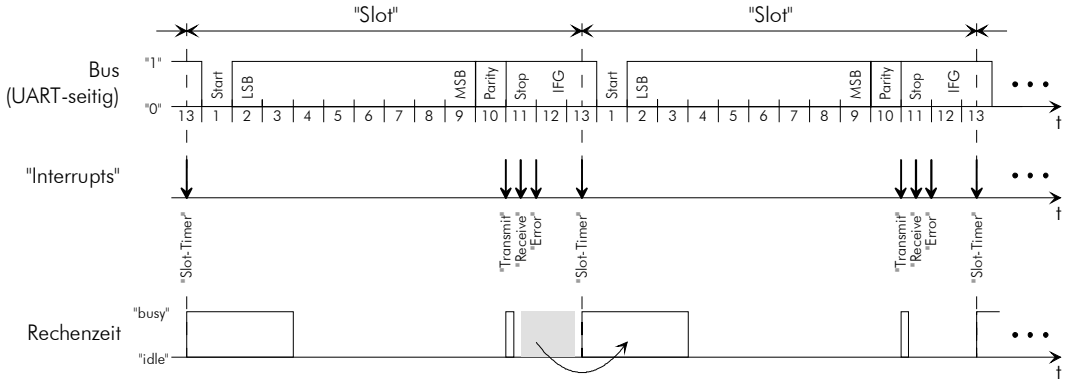


Bild 4.12: Rechenzeiten bei einer rechenzeitorientierten „Task“-Aufteilung

Durch den gewonnenen Spielraum am „Slot“-Ende entspannt sich die Konfliktsituation erheblich. Wenn man die Problematik der Zeitfehler einmal außer Acht lässt, kann die CPU jetzt sehr stark mit „Basic-TTP/A“ ausgelastet werden. Theoretisch darf die Haupt- bzw. „Slot-Task“ bis zum „Slot“-Ende reichen. Dazu muss jedoch sichergestellt sein, dass kein empfangenes Byte oder kein „Error-Flag“ überschrieben wird. Eine Kollision zwischen der Haupt- und „Transmit-Task“ ist unproblematisch, falls die „Transmit-Task“ den Vorrang hat. Da die „Slot-“ und „Transmit-Task“ getrennte Betriebsmittel haben, können keine Ressourcenkonflikte oder Inkonsistenzprobleme entstehen. Eine Verzögerung der „Slot-Task“ ist, aufgrund der Kürze der „Transmit-Task“, nicht möglich oder sehr unwahrscheinlich. Zum einen ist der Spielraum zwischen dem „Transmit-“ und dem „Slot-Timer-Interrupt“ bei max. zulässigen, gleichgerichteten Zeitfehlern i.A. nicht Null (Minimum ca. eine halbe Bitzeit). Zum anderen werden die Toleranzen in dieser Situation nicht voll ausgereizt, da die Baudrate hier so hoch ist, dass kein RC- oder „On-Chip“-Oszillator mehr die notwendige Taktfrequenz liefern kann. Und schließlich gefährdet eine Unterbrechung der „Slot-Task“ durch die „Transmit-Task“ keine zeitkritischen „Basic-TTP/A“-Abläufe, weil diese im Anfangsbereich der „Slot-Task“ stattfinden müssen.

Unter alleiniger Beachtung der Rechenzeitverhältnisse bestünde theoretisch also die Möglichkeit, die CPU voll auszulasten. Im Gegensatz zum vorherigen Ansatz, mit einer max. Auslastung von 5..10 %, bedeutet das eine Erhöhung der max. Baudrate um den Faktor 10..20. Damit verschiebt sich Grenze vom unteren bis mittleren 10 kBit/s-Bereich in den oberen 100 kBit/s. Praktisch macht eine 100 %-ige Auslastung der CPU durch „Basic-TTP/A“ allerdings keinen Sinn, weil die Applikation dann nicht mehr rechnet – es versteht sich von selbst, dass „Basic-TTP/A“ höher priorisiert ist. Deshalb ist eine Beschränkung der max. Baudrate auf den mittleren 100 kBit/s-Bereich bezüglich der Auslastung einer CPU mit ca. 10 MIPS sinnvoll und i.A. vollends ausreichend.

Dass der rechenzeitorientierte Ansatz zum Erfolg geführt hat, sollen die Oszillogramme in Bild 4.13 beweisen. Sie zeigen die Rechenzeiten (oben) und Busverhältnisse (unten) eines „Basic-TTP/A-Mas-

ters“ mit C16x-µC bei 125 kBit/s (rechts) und 625 kBit/s (links). Für die Messung der Rechenzeit und die „Triggerung“ war es nötig, den Code zu instrumentieren, weshalb die CPU-Belastung ein klein wenig größer ausfällt. Die Bussignale dienen zur Orientierung und zum Vergleich. Ohne sie wäre eine Runden- oder „Slot“-Zuordnung und ein Bezug auf die Bitzeiten nicht bzw. kaum möglich. Im Rechenzeitsignal stammen die breiten Impulse von der „Slot-Task“ und die schmalen Impulse von der „Transmit-Task“. Letztere ist aufgrund des verwendeten „Tristate-Physical-Layers“ (RS485) erforderlich. An den schmalen Nadelimpulsen erkennt man übrigens, dass der „Master“ im ersten „Slot“ („Fireworks-Frame“) und im dritten „Slot“ (irgendein Daten-„Frame“), der drei „Slot“ langen Runde, gesendet hat.

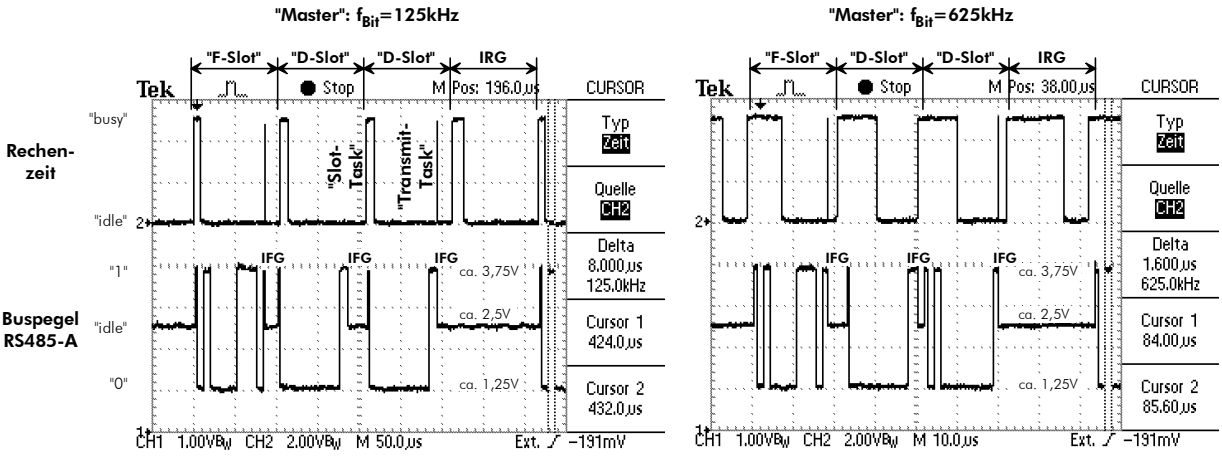


Bild 4.13: Beispiele für die Rechenzeiten eines „Basic-TTP/A-Masters“ mit C16x-µC

Gemäß Bild 4.13, beträgt die Auslastung der CPU durch den „Basic-TTP/A-Master“ bei 125 kBit/s ungefähr 10 % und bei 625 kBit/s ungefähr 50 % – eine Vermessung ergab 10,7 % und 53,6 %[99]. Da Letztere die höchste Baudrate des C16x-µCs ist (f_{CPU} = 20 MHz), kann die auslastungsbedingte, obere Baudrate nur geschätzt werden. Im rechten Oszillogramm sieht man besonders deutlich, dass der kritische Bereich in der IRG liegt. Dort ist die Dauer der „Slot-Task“ im „Master“ mit ca. 14 µs am größten und dem zufolge der Spielraum mit ca. 6 µs am kleinsten; die Gründe hierfür folgen in den nächsten Abschnitten. Aus diesen Werten errechnet sich die theoretisch auslastungsbedingte, obere Baudrate der C16x-µCs zu 625 kBit/s·(1 + 6 µs/14 µs) ≈ 893 kBit/s. Mit einer Code-Optimierung – die Implementierung ist in C gehalten – wären ungefähr 1 MBit/s denkbar. Diese Baudraten würden allerdings eine CPU-Last von ca. 76,6 % bzw. 85,8 % produzieren, und das wäre nicht sinnvoll. Schließlich ist „Basic-TTP/A“ das Mittel und nicht der Zweck.

Aus Versuchen, über die in Kapitel 5 berichtet wird, ist bekannt, dass anspruchsvolle oder aufwändige Applikationen, wie z.B. eine schnelle adaptive Regelung oder ein „IFS-Gateway“, bei 625 kBit/s mit C16x-µCs zwar noch funktionieren, niederpriore Aufgaben, wie z.B. eine Sollwertverstellung oder Prozessdatenanzeige, aber bereits träge ablaufen[100]. Einfache Applikationen, die für viele Sensoren und Aktoren ausreichen, funktionieren hingegen tadellos. Deshalb sollte die Belastung der CPU durch „Basic-TTP/A“ nicht wesentlich höher als 50 % sein – dass die max. Baudrate des C16x-µCs in die-

[99] In einem „Slave“ betragen die gemessenen CPU-Lasten bei denselben Baudraten ≈ 4,8 % und ≈ 23,8 % (grob die Hälfte), weil er sich nicht um die Rundensteuerung kümmern muss (siehe Abschnitt 5.2.6).

[100] Diese Applikationen sind für 125 kBit/s ausgelegt. Bei dieser Baudrate, die viele Sensor/Aktorbusse bereits nicht mehr unterstützen, merkt man von „Basic-TTP/A“ auf den ca. 10 MIPS starken C16x-µCs praktisch nichts.

sem Bereich liegt, ist Zufall. Ein Vorteil dieser „weichen“ Restriktion ist eine einfachere Implementierung. Wie Bild 4.13 zeigt, können bei ca. 50 % Auslastung des µCs „Slot-“ und „Transmit-Task“ nicht kollidieren. Dadurch ist keine „preemtive“ Verwaltung der „Basic-TTP/A-Tasks“ notwendig. Es reicht aus, wenn sie mit der höchsten Priorität bearbeitet werden, um sich von der Applikation abzuheben. Auf diese Weise können die „Basic-TTP/A-Tasks“ auch als einfache ISRs realisiert werden, was insbesondere der „Slave“-Implementierung sehr entgegenkommt. „Slaves“ sollen ja vorwiegend mit „low-cost“ µCs arbeiten, deren Ressourcen bekanntlich nicht üppig sind. Für einen zusätzlichen, kleinen „Realtime-Kernel“ ist oft kein Platz mehr (siehe Kapitel 5).

Weitere Vorteile ergeben sich aus der geringeren „Task“-Anzahl, des rechenzeitorientierten Ansatzes. Im Minimalfall benötigt man lediglich einen „Timer-“ und „Receive-Interrupt“ von einem Hardware-UART oder einer Emulation, womit die wenigsten „low-cost“ µCs ein Problem haben. Außerdem wird der Speicherbedarf reduziert. Zum einen sind weniger „Entry/Exit“-Codes für die ISRs und gegebenenfalls weniger „Stacks“ für die „Tasks“ erforderlich. Und zum anderen entfällt eine „Intertask“-Kommunikation, wegen der Unabhängigkeit der „Slot-“ und „Transmit-Task“. Darüber hinaus schont die Absenz einer „Intertask“-Kommunikation und die geringere Anzahl der „Task“-Wechsel die Ressource Rechenzeit. Summa summarum bietet eine rechenzeitorientierte „Task“-Aufteilung wesentlich bessere Voraussetzungen für eine Implementierung von „Basic-TTP/A“ als eine ereignisorientierte „Task“-Aufteilung. Das gilt vor allem für die Zielarchitektur „low-cost“ µC. Es zeigt sich also, dass ein gutes Konzept viel wichtiger ist, als eine (händisch) optimierte Implementierung.

4.2.3 „Interleave“-Algorithmus

Die rechenzeitorientierte „Task“-Aufteilung ist eine notwendige Bedingung für hohe Baudraten. Sie kann hohe Baudraten allerdings nicht garantieren, da sie keine Aussage über die Zeitfehlerproblematik aus Abschnitt 4.1 macht. Deshalb spielt der Algorithmus eine ebenso wichtige Rolle. Er stellt die hinreichende Bedingung für hohe Baudraten dar und berücksichtigt die Zeitfehlerproblematik. Hier wird bewusst der Singular verwendet, weil das nur die „Slot-Task“ betrifft; der Algorithmus in der „Transmit-Task“ ist trivial. Kern der „Slot-Task“ ist, entsprechend Bild 4.14, ein so genannter „Interleave“-Algorithmus. Seinen Namen hat Algorithmus aufgrund der überlappenden bzw. verschränkten Bearbeitung der „Slot“-Aufgaben[101]. Das ist zwar teils auch eine Folge der rechenzeitorientierten „Task“-Aufteilung, jedoch hauptsächlich auf den „Interleave“-Algorithmus zurückzuführen, wie die Ausführungen in diesem Unterpunkt zeigen werden.

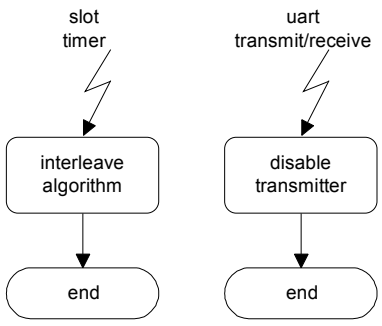


Bild 4.14: Rechenzeitorientierte „Task“-Aufteilung

[101] interleave (engl.) = überlappen, verschränken; man findet diesen Begriff nur in einem technischen Wörterbuch.

Prinzip

Für die Erklärungen ist das Prinzip des „Interleave“-Algorithmuses in Form eines PAPs (Programmablaufplan) in Bild 4.15 skizziert. Auf Details wird geflissentlich verzichtet. Sie würden erstens die Darstellungsmöglichkeiten überfordern, zweitens die Übersichtlichkeit beeinträchtigen und drittens eine Unterscheidung zwischen „Master“ und „Slave“ notwendig machen. Der interessierte Leser mag sie im dokumentierten Quellcode auf der beiliegenden CD im Anhang nachlesen.

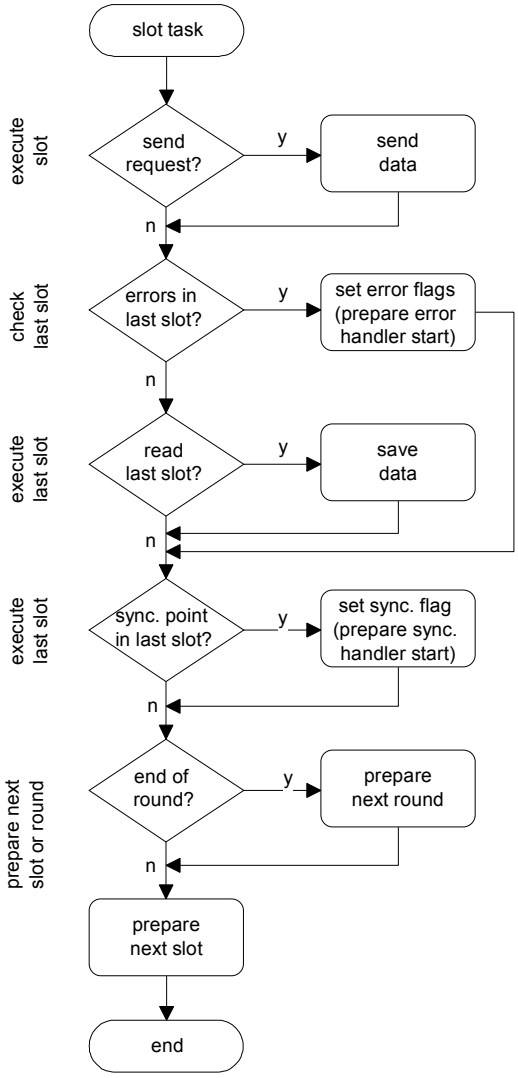


Bild 4.15: Prinzip des „Interleave“-Algorithmuses

Bekanntlich wird die „Slot-Task“ zu Beginn eines „geschedulten Slots“ aktiviert. Falls es sich um einen Schreib-„Slot“ handelt, leitet die „Slot-Task“ unverzüglich das Versenden des entsprechenden Datums in die Wege. Das ist allerdings nur möglich, wenn der „Slot“ vorbereitet wurde. Nun zeigt Bild 4.15, dass diese Aufgabe am Ende der „Slot-Task“ erledigt wird. Folglich ist eine Aktivierung der „Slot-Task“ vor dem jeweiligen „Slot“ notwendig. Bei einem schreibenden Buszugriff sollte das am besten im vorangehenden „Slot“ geschehen, weil zur Vorbereitung auch das Bereitlegen des entsprechenden Datums gehört und man in der Regel aktuelle Daten haben möchte. Hie und da ist es auch sinnvoll, gewisse Verzögerungen in Kauf zu nehmen. Wenn z.B. ein Sensorknoten in mehreren

„Slots“ einer Runde teilnimmt und der Abstand dazwischen im Verhältnis zur Sensorträgheit vernachlässigbar ist, sind zusätzliche Aktivierungen der „Slot-Task“ unnötig. Auf diese Weise kann man die Belastung der CPU durch „Basic-TTP/A“ reduzieren.

Nachdem ein Knoten diese zeitkritische Aufgabe gemäß seiner RODL erledigt hat, folgt eine Fehlerprüfung für den letzten „Slot“. Weil dieser „Slot“ bereits in der Vergangenheit liegt, können sämtliche Fehler auf einmal analysiert werden. Das ist ein weiterer Vorteil gegenüber dem ereignisorientierten Ansatz aus Abschnitt 4.2.2. „Basic-TTP/A“ kann, neben den drei UART-Fehlern „Parity-“, „Overrun-“ und „Frame-Error“, zusätzlich die folgenden fünf Ablauf- und Vergleichsfehler erkennen: „Receive-Not-Send“, „Missing-Frame“, „Missing-Fireworks“, „Bad-Fireworks“ und „Frame-In-IRG“; Näheres befindet sich im Quellcode. Eine verteilte Erkennung dieser Fehler wäre aufwändiger, umständlicher und anfällig gegen Inkonsistenzen. Im Fehlerfall werden von der „Slot-Task“ die entsprechenden „Error-Flags“ gesetzt und gegebenenfalls die nötigen Schritte zum Starten eines „Error-Handlers“ unternommen. Die Reaktion darf allerdings nicht in der „Slot-Task“ erfolgen. Sie muss auf einer niedrigeren Prioritätsstufe geschehen, weil die Fehlerbehandlung zum großen Teil von der Applikation abhängt und damit zeitlich nicht kalkulierbar ist. Ansonsten besteht die Gefahr einer Fehlerfortpflanzung durch verletzen von „Basic-TTP/A-Deadlines“. Je nach Ressourcen des µCs, kann die Fehlerreaktion in der Applikation per „Polling“ oder per „Error-Handler“ geschehen. Ersteres ist der einfachste Fall, reicht für viele Applikationen aus und funktioniert auf jedem „low-cost“ µC. Letzteres setzt entweder ein Realzeitbetriebssystem oder einen freien „Interrupt“ voraus. Die C16x-Portierung aus Kapitel 5 verwendet z.B. den „Error-Interrupt“ des UARTs mit einer niedrigen Priorität dafür. µCs ohne priorisierbare „Interrupts“ müssen in der entsprechenden ISR lediglich das „Interrupt-Enable-Flag“ setzen, um das Gleiche zu erreichen[102].

Sofern die Fehlerprüfung negativ ausfällt und der letzte „Slot“ ein Lese-„Slot“ war, speichert die „Slot-Task“ das empfangene Byte ab. Den Speicherort oder Informationen darüber entnimmt die „Slot-Task“ der RODL. Bei einer positiven Fehlerprüfung wird dieser Vorgang Bild 4.15 zufolge übersprungen. Damit ist das empfangene Byte aber nicht verloren. Die Applikation oder ein „Error-Handler“ kann es bis zum Empfang des nächsten „Frames“ lesen und z.B. nach einer FEC oder Filterung trotzdem verwenden. Falls kein „Frame“ empfangen wurde, kann z.B. eine Extrapolation eine Lösung sein.

Der nächste Schritt im PAP, das Setzen eines „Synchronization-Flags“ bzw. das Vorbereiten des Starts eines „Synchonization Handlers“, ist optional. Er gestattet eine Synchronisation zwischen „Basic-TTP/A“ und der Applikation in einem Knoten. Das ist vor allem in Anwendungen mit höheren Ansprüchen, wie z.B. das gleichzeitige Reagieren mehrerer Knoten auf einen „Broadcast“ oder die zeitliche Steuerung verteilter Anwendungen, vorteilhaft. Im Demonetz (siehe Kapitel 5) werden auf diese Weise eine Notausfunktion realisiert, ein Abtastregelkreis über den Bus geschlossen, Inkonsistenzen bei der Übertragung von Mehrbytedaten verhindert und eine verteilte printf-Funktion synchronisiert. Des Weiteren ermöglichte der Synchronisationsmechanismus eine unproblematische Erweiterung von „Basic-TTP/A“ um die ersten Teile des IFS-Bausteins (siehe Abschnitt 3.5 und 5.4.3). „Basic-TTP/A“ (Schicht 2) arbeitet mit Daten und das IFS (Schicht 7) mit Nachrichten. Für Daten, insbesondere für Prozessdaten, ist eine zyklische Übertragung typisch, wohingegen Nachricht eher sporadisch auftreten. Aus diesem Grund ist es sinnvoll, Nachrichten in eine Extrarunde zu verpacken. Diese Nachrichtenrunde wird nur bei Bedarf initiiert, also sporadisch, um keine Übertragungskapazität zu vergeuden. Der Synchronisationsmechanismus erwies sich insofern nützlich, als er die Identifizierung und Bearbeitung der Nachrichten vereinfacht und Rechenzeit spart. Bezüglich der Implementierung gilt das Gleiche wie für die zuvor beschriebene Fehlerbehandlung.

[102] Aus Gründen der Portierbarkeit arbeitet die Implementierung mit einer „Error-Hook-Function“ in der Applikation.

Am Ende der „Slot-Task“ wird der nächste „Slot“ oder, falls kein weiterer „Slot“ in der laufenden Runde von Bedeutung ist, die IRG vorbereitet. Die Vorbereitung auf die nächste Runde geschieht in der IRG – aus diesem Grund dauert die „Slot-Task“ in der IRG etwas länger (siehe Oszillogramme in Bild 4.13). Früher ist das im „Master“ nicht sinnvoll und im „Slave“ nicht möglich. „Basic-TTP/A“ besitzt mit der Rundensteuerung einen ereignisorientierten Anteil. Damit kann die Applikation im „Master“ auf gewisse Geschehnisse flexibel reagieren. Je nach Lage, teilt sie dem „Master“ rechtzeitig mit, welche Runde er als nächstes initiieren soll. Die „Deadline“ hierfür ist die IRG. Eine vorzeitige Festlegung der nächsten Runde würde das Reaktionsvermögen des „Masters“ verschlechtern. Es könnte ja sein, dass kurz vor der IRG noch etwas Wichtigeres passiert. Natürlich gäbe es Lösungen, um diesen Fall abzufangen. Sie wären jedoch komplizierter und aufwändiger als die Vorgestellte. Außerdem sollte der „Master“, als kontrollierende Instanz, ohnehin an jedem „Slot“ teilhaben. Nur so kann er fehlende oder verfälschte „Frames“, deren Ursachen z.B. ein „Slave“-Ausfall oder Buskollisionen sein können, erkennen und gegebenenfalls angemessen darauf reagieren. In einem „Slave“ muss die Rundenvorbereitung in der IRG stattfinden. Schließlich wartet er danach auf ein „Fireworks-Frame“ und würde bei jedem Daten-„Frame“ einen Fehler auslösen.

Notwendigkeit

Das Besondere am „Interleave“-Algorithmus ist die unverzügliche Reaktion am Anfang. Sie ermöglicht zum einen „Jitter“-freie „Fireworks-Frames“ bei Verwendung von Hardware-UARTs und zumindest „Jitter“-arme „Fireworks-Frames“ bei Verwendung von Emulationen; in Abschnitt 4.2.4 wird eine entsprechende Methode für Hardware-UARTs vorgestellt. Zum anderen leistet sie einen wesentlichen Beitrag für die Kompensierbarkeit des „Slot-Timer“-Lagefehlers. Für die Erklärung wird ein Vergleich mit einer anderen Bearbeitungsreihenfolge durchgeführt: Wenn die „Slot“-Vorbereitungen oder anderweitige Aufgaben am Anfang der „Slot-Task“ stünden, verginge mehr Zeit bis zum Sendebefehl – die Reihenfolge danach ist weniger wichtig. Theoretisch würde die „Slot-Timer“-Lagefehlerkorrektur diesen Zeitversatz automatisch kompensieren. In der Praxis gibt es jedoch Begrenzungseffekte. Mit zunehmender Baudrate werden die Reaktionszeiten kürzer. Die Ausführungszeiten bleiben jedoch konstant, weshalb sich die Spielräume verkleinern. Wenn keine Spielräume mehr existieren, ist die Grenze der „Slot-Timer“-Lagefehlerkorrektur erreicht, da man keine negativen Zeiten einstellen kann. Ein ähnliches Problem hat auch der „Master“. Für „Jitter“-freie „Fireworks-Frames“ muss er einen Phasenabgleich zwischen seinem „Slot-Timer“ und UART durchführen (siehe Abschnitt 4.2.4). Unter anderem spielt dabei das Kompensieren von Ausführungszeiten eine Rolle. Je größer also der erwähnte Zeitversatz ist, desto kleiner wird die max. Baudrate.

Außerdem würden die nötigen Programmverzweigungen bei einer anderen Bearbeitungsreihenfolge viele und unterschiedlich lange Programmpfade erzeugen. Die Folge wäre eine zusätzliche Varianz der Ausführungszeit bis zum Sendebefehl. Schließlich werden die Entscheidungen zur Laufzeit getroffen. Aus Abschnitt 4.1 ist bekannt, dass man diese zufälligen Zeitfehler nur puffern kann. In ungünstigen Zuständen, die vorzugsweise bei hohen Baudraten auftreten können, ist dafür aber kein Platz mehr in der IFG. Folglich würde eine andere Bearbeitungsreihenfolge bzw. ein anderes Programmkonzept eine größere IFG voraussetzen. Des Weiteren besteht die Gefahr, dass die zusätzliche Varianz einen unzulässigen „Fireworks-Jitter“ verursacht. Die Methode in Abschnitt 4.2.4 toleriert zwar gewisse Schwankungen, wenn jedoch ein Grenzwert überschritten wird, beginnt das „Fireworks-Frame“ zu springen. Beim „Interleave“-Algorithmus ist der konstante Zeitversatz hingegen gering und die verzweigungsbedingte Varianz Null. Erstens ist der Programmpfad bis zum Sendebefehl minimal kurz und zweitens wird immer der gleiche Programmpfad durchlaufen. Der „Interleave“-Algorithmus ist sicherlich nicht die einzige Lösung. Bisher liefert er jedoch die besten und vor allem die ersten brauchbaren Ergebnisse.

Nachteile

Ein Problem des „Interleave“-Algorithmuses ist der Start. Die verschränkte Arbeitsweise, macht ein explizites Vorbereiten des ersten „Slots“ erforderlich. Hierzu muss im „Master“ der letzte Teil der „Slot-Task“ während des „Bootens“ separat abgearbeitet werden. Erst dann ist eine Aktivierung der „Slot-Task“ zulässig. Diesen etwas komplizierteren „Boot“-Vorgang kennt ein „Slave“ nicht. Stattdessen muss er den „Interleave“-Algorithmus mit jedem „Fireworks-Frame“ neu aufsetzen. Daraus entstehen allerdings keine Nachteile. Für einen „Slave“ ist der Rundenstart, aufgrund des ereignisorientierten Anteils in „Basic-TTP/A“, nicht planbar. Unabhängig vom Algorithmus bleibt ihm also gar nicht anderes übrig, als nach dem „Fireworks-Frame“ den ersten „Slot“ vorzubereiten. Von der schwierigeren und etwas umfangreicheren Bearbeitung des „Schedules“ sind hingegen beiden Knotentypen betroffen. Im Vergleich zu einer nicht überlappenden Arbeitsweise, muss beim „Interleave“-Algorithmus ein „Slot“ vorbereitet, abgearbeitet und nachbereitet werden. Entsprechend komplex fällt die Implementierung aus. Weitere Nachteile sind nicht bekannt.

4.2.4 „UART/Fireworks-Jitter“

Phänomen

Die serielle Kommunikation mit UARTs ist seit langem bekannt und weit verbreitet. Also meint man annehmen zu dürfen, dass über diese Bausteine alles bekannt ist. Wie sich jedoch herausstellte, machte eine nicht bzw. nur indirekt dokumentierte Eigenschaft von UARTs erhebliche Probleme bei der Umsetzung von „Basic-TTP/A“. Es handelt sich um den mehrfach erwähnten UART-„Jitter“, einem zeitlichen Indeterminismus beim Senden[103]. Das Phänomen ist in Bild 4.16 zu sehen. In diesem Beispiel wurde das „Transmit“-Register eines UARTs periodisch beschrieben. Wider Erwarten beginnt der UART nicht immer sofort zu senden. Der „Frame“ liegt irgendwo in dem grauen Bereich des Oszillogramms. Aus dessen Breite kann entnehmen, dass der Verzug nach dem Schreibvorgang auf das „Transmit“-Register bis zu einer Bitzeit lang ist; mit „Critical-Sections“ o.Ä. lässt sich das nicht mehr erklären. Für „normale“ Anwendungen, wie z.B. einer simplen PC-Kommunikation, ist dieser Indeterminismus unerheblich. Dort müssen die „Frames“ nicht exakt platziert werden, so wie in „Basic-TTP/A“. Und offensichtlich hat man bei der Konstruktion von UARTs darauf keinen Wert gelegt.

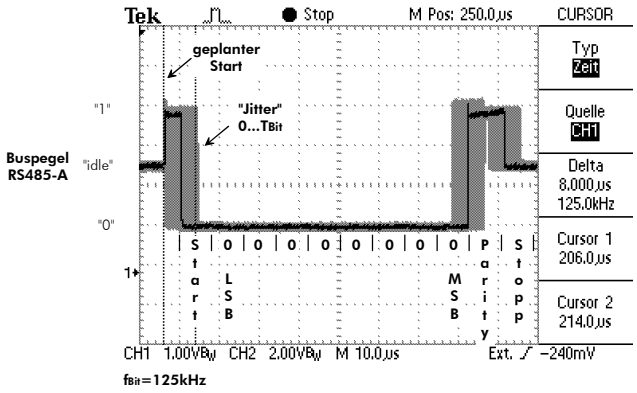


Bild 4.16: Phänomen UART-„Jitter“ (grauer Bereich)

[103] Selbst namhafte Hersteller wissen bzw. wussten davon nichts.

Ursache

UARTs werden mit einer 8- oder 16-fach höheren Frequenz als der Baudrate getaktet. Mit diesem Takt tasten sie die eingehenden Bits ab und bestimmten die Bitmitten und Startbitflanken. Außerdem legen sie damit die „Interrupt"-Zeitpunkte fest. Den Bittakt erzeugen die UART-Bausteine mit internen Frequenzteilern. Sie besitzen in der Regel je einen für den Sende- und Empfangsteil, um eine Vollduplex-Kommunikation zu unterstützen. Den prinzipiellen Aufbau eines UARTs zeigt Bild 4.17. Es wurde der Teilerfaktor 16 gewählt, da er wesentlich häufiger vorkommt als der Teilerfaktor 8 (siehe Abschnitt 8.9). Auffällig ist die höhere Komplexität des Empfängers. Im Gegensatz zum Sender, der nur agiert, muss der Empfänger reagieren und zwar rechtzeitig. Dazu setzt er unmittelbar nach einer Startbitflanke seinen Teiler („Reset") zurück und startet den Empfangszyklus („Start"). Anschließend beginnt der Teiler zu zählen und liefert bei jedem Halbmodulo das Signal „Bithalf" für die Bitmittenabtastung – bei einer Mehrfachabtastung sind das entsprechend viele Signale – und bei jedem Modulo das Signal „RxClock" für die Bitende-Erkennung. Je nach Einstellung, beendet die Steuerung nach neun bis zwölf Bits den Empfang und aktiviert wieder die Startbiterkennung.

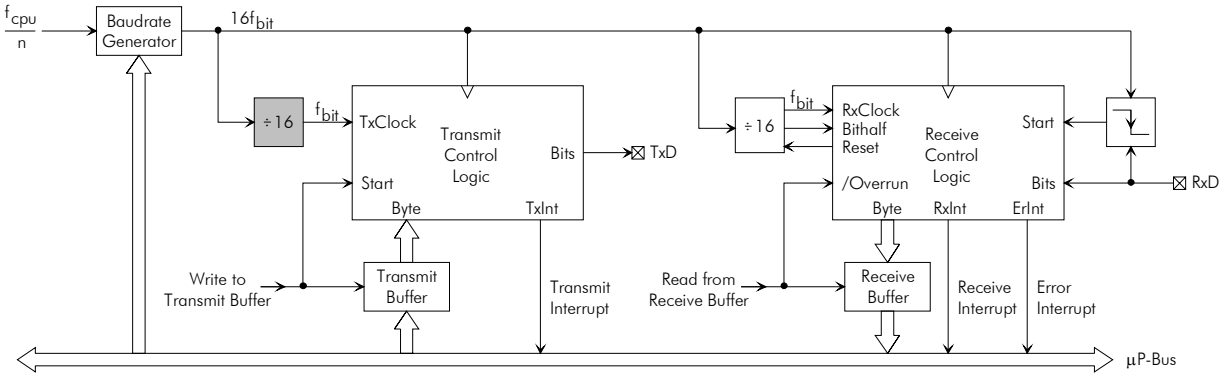


Bild 4.17: Blockschaltbild eines UARTs mit den wichtigsten Elementen

Das Senden ist deutlich einfacher. Nach dem Beschreiben des „Transmit"-Registers bzw. „Transmit-Buffers" kopiert die Sendesteuerung den Inhalt in ein Schieberegister. Dieses wird im Anschluss mit dem Bittakt „TxClock" verbunden und legt bei jedem Modulo des grau unterlegten Teilers das jeweils nächste Bit, zuzüglich der „Frame"-Bits, auf den Pin „TxD". Eine Rücksetzbarkeit des Teilers im Sender ist nicht nötig. Schließlich muss sich der Empfänger auf ihn synchronisieren und nicht umgekehrt. Aus diesem Grund enthält so gut wie kein UART diese Möglichkeit – dem Autor ist lediglich eine Ausnahme bekannt[104]. Und genau das ist die Ursache für den UART-„Jitter". Dessen Entstehung soll anhand von Bild 4.18 erklärt werden. Wenn man den UART, im Zeitraster des µCs, kurz vor dem Modulo beschreibt, entsteht praktisch kein Sendeverzug. Macht man das Gleiche danach, muss man auf den nächsten Modulo warten. So kann die Verzögerung, je nach Zeitpunkt des Beschreibens, bis zu einer Bitzeit betragen. Für „Basic-TTP/A" stellt das insofern ein Problem dar, als sich ein Knoten mit seinen Buszugriffen nicht am UART, sondern am „Slot-Timer" orientieren muss. Selbst wenn man den Zählerstand des Teilers kennen würde[105], könnte man das Problem nicht lösen. Das Einzige was man dann wüsste, wäre der jeweilige, meistens variable Sendeverzug der „Frames". Die Tatsache des Sendeverzugs bliebe allerdings unberührt. Interessanterweise machen sich einige UARTs bzw. µCs,

[104] Es handelt sich um den 32 Bit µC AT91M40400 von Atmel. Sein UART setzt beim Beschreiben des RSTTX-Bits im UART-„Control-Register" die komplette „Transmitter"-Logik zurück [120].

[105] Da dieser Teiler weder lesbar noch beschreibbar ist und außerdem sein „Reset"-Zustand nur vermutet werden kann, wäre dafür viel Aufwand nötig.

wie z.B. der C16x, diesen Umstand für eine effiziente Übertragung zu Nutze. Sie verwenden den „Transmit-Interrupt“, falls vorhanden, um den „Transmit-Buffer“ im Stoppbit nachzuladen. In der Regel kann der UART bzw. µC damit ohne eine Pause das nächste „Frame“ senden.

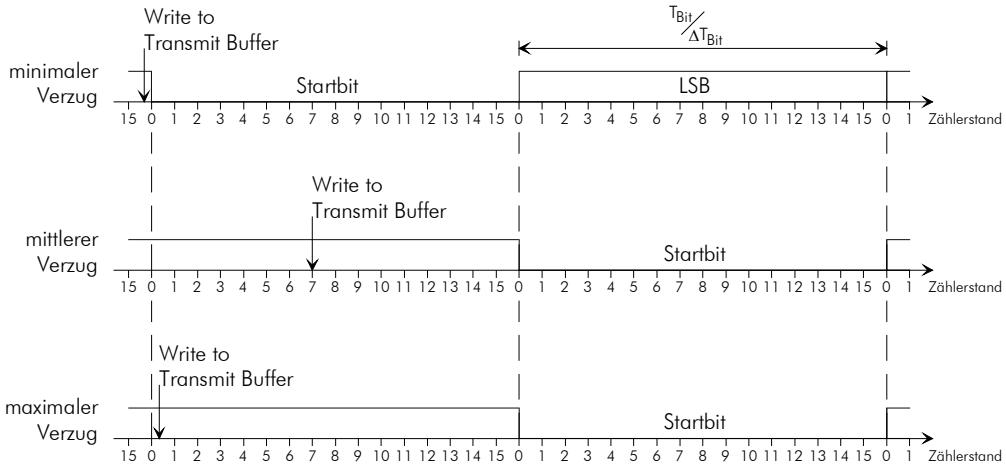


Bild 4.18: Entstehung des UART-„Jitters“

Phasenabgleich

Es ist nicht schwer zu erahnen, dass dieses Problem für „Basic-TTP/A“ soweit als nötig gelöst wurde. Allerdings darf man die anschließende Lösung nicht isoliert oder als zufällig betrachten, da in einem iterativen Prozess die Voraussetzungen dafür geschaffen wurden. Es war erforderlich die Funktionsweise und das Konzept von „Basic-TTP/A“ so lange zu modifizieren, bis dieses Problem gelöst werden konnte, ohne neue Probleme zu schaffen oder Lösungswege anderer Probleme zu verbauen. Aus diesem iterativen Prozess stammen die feste Kopplung zwischen dem „Slot-Timer“ und UART im „Master“ ($T_{Slot,\,M} = 13 \cdot T_{Bit,\,M}$), die Gleichheit von IRG- und „Slot“-Dauer und teils auch der „Interleave“-Algorithmus. Diese Maßnahmen bilden die Grundlage für „Jitter“-freie „Master-Frames“ und damit auch für die geforderten „Jitter“-freien „Fireworks-Frames“. Ein „Slave“ profitiert davon leider nicht. Aufgrund seiner Uhrenfehler, ist i.A. $T_{Slot,\,S}$ ungleich $T_{Slot,\,M}$ und weder $T_{Slot,\,S}/T_{Slot,\,M}$ noch $T_{Slot,\,S}/T_{Bit,\,S}$ konstant. Deshalb werden seine „Frames“, bezogen auf die „Slot“-Grenzen des „Masters“, immer „jittern“.

Die Gleichheit von IRG- und „Slot“-Dauer ermöglicht dem „Master“ äquidistante Schreibzugriffe auf das „Transmit“-Register. Mit der festen Kopplung zwischen dem „Slot-Timer“ und UART wird hingegen sichergestellt, dass die Schreibzugriffe beim gleichen Zählerstand des Sendeteilers erfolgen. Auf diese Weise haben die „Master-Frames“ theoretisch immer den gleichen Abstand zu den „Slot“-Grenzen. Theoretisch deshalb, weil unterschiedliche lange Programmpfade in der „Slot-Task“ und „Critical-Sections“ in der Applikation die Äquidistanz der Schreibzugriffe stören können. Außerdem besteht die Möglichkeit, dass zeitvariante Befehle ebenfalls dazu beitragen. Schließlich ist der „Master“ der ressourcenstärkster Knoten in einem „Basic-TTP/A“-Netz. Sein µC, DSP oder µP kann durchaus über eine „Pipeline“, „Buffers“, „Caches“, eine „Branch Prediction“, eine „Speculative Execution“ oder eine „Out of Order Execution“ verfügen.

Zur Reduktion der möglichen Zeitvarianz trägt der „Interleave“-Algorithmus ein gutes Stück bei, indem er das Durchlaufen unterschiedlicher Programmpfade bei den zeitkritischen Aktionen eines Schreib-„Slots“ verhindert. Auf diese Weise entstehen keine algorithmusbedingten und weniger CPU-

bedingte Zeitschwankungen. Ersteres ist einsichtig, dagegen bedarf Letzteres ein paar Erläuterungen. Wenn stets der gleiche Programmpfad bearbeitet wird und keine Unterbrechungen stattfinden, machen die erwähnten Beschleunigungsmechanismen stets dasselbe, da jeder Befehl stets die gleiche Vorgeschichte hat. „Pipeline-Stalls“ oder „Caches-Misses“ treten immer an der gleichen Stelle auf, sofern sie auftreten. Außerdem sind Sprungvorhersagen oder vorgreifende Befehlsverarbeitungen entweder richtig oder falsch. Sie ändern sich jedoch nicht. Folglich würden keine Zeitschwankungen entstehen, träfen alle Annahmen für den „Master“ zu. Allerdings ist das nicht realisierbar, weil der „Master“ zur Laufzeit Entscheidungen treffen muss. Im Vergleich zu einem Algorithmus mit wechselnden Programmpfaden, sorgt der „Interleave“-Algorithmus aber immerhin dafür, dass die Annahmen teilweise erfüllt werden, wodurch die Zeitschwankungen geringer ausfallen.

Trotzdem kann die verbleibende Zeitunsicherheit einen „Fireworks-Jitter“ bewirken. Wenn der Schreibzugriff kurz vor dem „Timer“-Überlauf stattfindet (Bild 4.18 oben), dann reicht eine kleine Verzögerung, z.B. durch eine „Critical-Section“ in der Applikation, aus, um das „Fireworks-Frame“ ein ganze Bitzeit springen zu lassen. Tückische daran ist, dass dieser Effekt nicht immer auftritt. Mit einer Applikation funktioniert es, mit einer anderen nicht. Manchmal genügt auch eine Codeoptimierung (Geschwindigkeit, Speicherplatz), um eine funktionierende Applikation in eine nichtfunktionierende zu verwandeln. Deshalb muss ein Respektabstand zu diesem kritischen Punkt eingehalten werden, was letztendlich auf einen Phasenabgleich zwischen „Slot-Timer“ und UART hinausläuft. Problematisch hierbei ist die Unsteuerbarkeit des Sendeteilers, weshalb eine direkte Methode als Lösung ausscheidet. Glücklicherweise wurde eine zuverlässige, indirekte Methode gefunden, dessen Prinzip mit Hilfe von Bild 4.19 erklärt werden soll. Jeder UART besitzt einen „Receive-Interrupt“ mit dokumentierten, relativen Auftrittszeitpunkt. Damit ist ein beobachtbares Ereignis bekannt, das einen Rückschluss auf den Zählerstand des Sendeteilers zulässt. Folglich ist der „Receive-Interrupt“ für einen Phasenabgleich zwischen „Slot-Timer“ und UART im „Master“ geeignet.

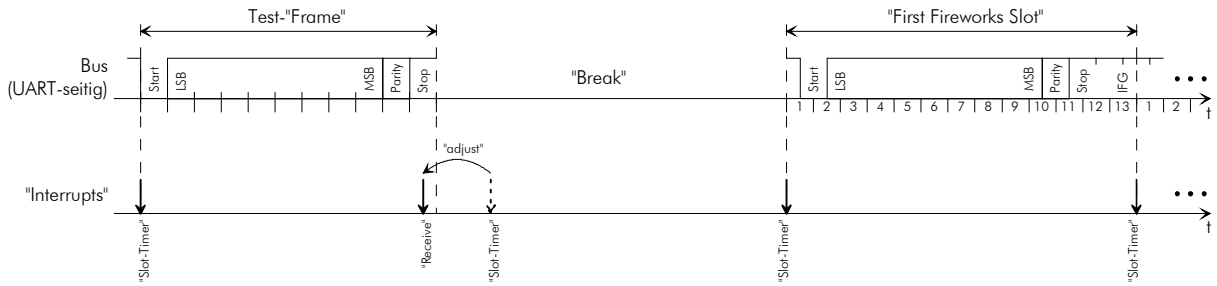


Bild 4.19: Prinzip des Phasenabgleichs zwischen „Slot-Timer“ und UART im „Master“

Das Vorgehen ist denkbar einfach. Während des „Bootens“ sendet der „Master“ ein Test-„Frame“. Er hört dieses Test-„Frame“ mit und wartet auf den „Receive-Interrupt“. Sobald dieser auftritt, justiert er die Phase seines „Slot-Timers“ auf eine geeignete Art und Weise. Es ist ratsam die Phase ungefähr auf die Bitmitte einzustellen. Erstens können die Schreibzugriffe auf den UART, wegen der erwähnten Beschleunigungsmechanismen, auch etwas früher auftreten als während des Abgleichvorgangs. Sollte die, vor allem durch „Caches“ verursachte, Varianz zu groß sein, kann man mit expliziten „Caches-Flushs“ oder einem „Cache-Lock“-Mechanismus eine Reduktion bewirken. Allerdings wird das so gut wie nie notwendig sein. Die betroffenen CPUs sind nämlich so schnell (>> 100 MIPS), dass eine Handvoll Befehle mehr oder weniger schnell ausgeführt, selbst bei Baudraten von 0,5..1 MBit/s, kaum ins Gewicht fallen. Zweitens genügt eine halbe Bitzeit Respektabstand zum Teilermodulo, um die hauptsächlich auftretenden Zeitverzüge durch „Critical-Sections“ und „Pipelines“ abzufangen. Aus

dem Unterpunkt 4.1 ist deren Dauer mit $\Delta T_{D,\,ISR} \approx 0..T_{Bit,\,M}/8$ bekannt. Es ist also genug Reserve für die eine oder andere Eventualität (z.B. für „Caches“) vorhanden. Drittens würde ein größerer Respektabstand die Reaktionszeit nach einem „Fireworks-Frame“ für die „Slaves“ unnötig verkürzen. Und viertens vereinfacht ein bitmittiger Phasenabgleich den Algorithmus, weil die „Receive-Interrupts“ typischerweise in der Mitte des Stoppbits, bzw. bei Mehrfachabtastungen kurz danach, erscheinen.

Realisierung

Die Realisierung eines universellen Phasenabgleichs erfolgt am besten über eine „Master“-spezifische Erweiterung der „Slot-Task“. Bild 4.20 zeigt den entsprechenden PAP. Er enthält einen gleich langen, parallelen Programmpfad bis zum zeitkritischen Sendepunkt (grau unterlegt). Beim „Booten“ wird dieser Programmpfad, durch Setzen des „Phase-Adjust-Flags“, Setzen des „Send-Request-Flags“ und Anstoßen der „Slot-Task“, einmal durchlaufen. Würde man die Erweiterung in den „Interleave“-Algorithmus einbauen, wäre die Vor- und Nacharbeit größer (siehe Bild 4.15).

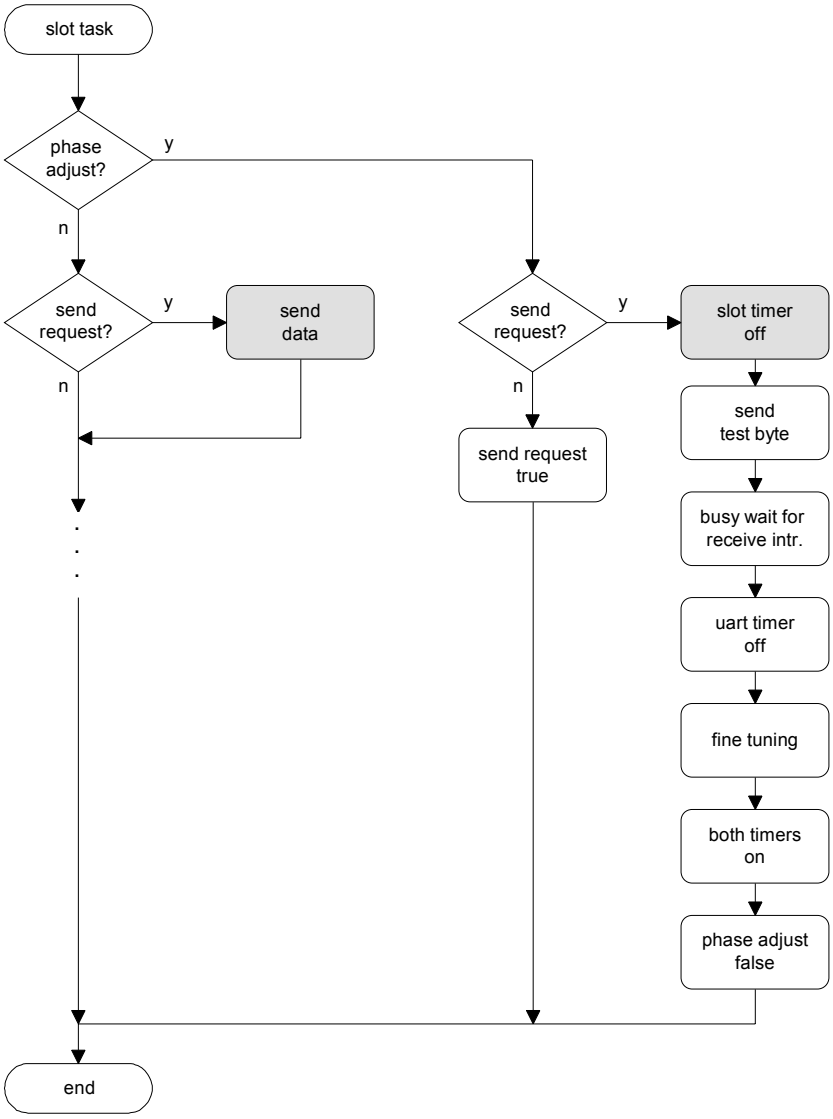


Bild 4.20: Erweiterung der „Slot-Task“ im „Master“ um den Phasenabgleich

Anstelle des Sendens, wird im Parallelpfad der „Slot-Timer“ angehalten. Damit friert man den Sendezeitpunkt ein, wodurch eine zeitliche Entkopplung stattfindet. Erst jetzt erfolgt das Senden des Test-„Frames“, mit anschließendem Warten auf das „Interrupt-Receive-Flag“. Aufgrund der kürzeren Reaktionszeit und der geringeren Ressourcen, ist es sinnvoll eine „Busy-Waiting-Loop, anstelle einer ISR zu verwenden.

Sofern möglich, wird der UART nach dem „Receive-Interrupt“ gestoppt. Ansonsten muss die darauf folgende Feineinstellung bei laufendem UART geschehen. Der dabei entstehende Fehler sollte bei hohen Baudraten korrigiert werden. Die Feineinstellung erfolgt durch addieren bzw. subtrahieren eines entsprechenden Wertes auf den bzw. vom eingefrorenen „Slot-Timer“-Wert. Sie ist Sache der Portierung und hat für das „Slot-Timing“ der „Slaves“ eine gewisse Bedeutung. Wenn das „Fireworks-Frame“ eine andere Lage hat als die „Slaves“ annehmen, entsteht ein zusätzlicher „Slot-Timer“-Lagefehler. Ohne Feineinstellung kann das z.B. beim Austausch eines „Masters“ mit Mehrfachabtastung und 16-facher Bitteilung durch einen „Master“ mit Mehrfachabtastung und 8-facher Bitteilung oder einen „Master“ ohne Mehrfachabtastung passieren. Deshalb ist es zweckmäßig, stets mit dem gleichen „Fireworks“-Verzug von z.B. $½ \cdot T_{Bit,\,M}$ zu arbeiten – es wäre nicht klug, würde man die „Slaves“ anpassen. Außerdem bietet die Feineinstellung eine Korrekturmöglichkeit für die seltenen UARTs mit „Receive-Interrupt“ am Stoppbitende.

Anschließend werden der „Slot-Timer“ und UART wieder gestartet und das „Phase-Adjust-Flags“ zurückgesetzt. Von nun an ist der „Master“ bereit, „Jitter“-freie und genau platzierte „Fireworks-Frames“ zu senden. Die zusätzliche Verzweigung am Anfang der „Slot-Task“ hat keine Auswirkung auf den „Interleave“-Algorithmus. Zum einen wird sie nach dem Abgleich immer gleich durchlaufen und zum anderen über die gleichenlangen Programmpfade kompensiert.

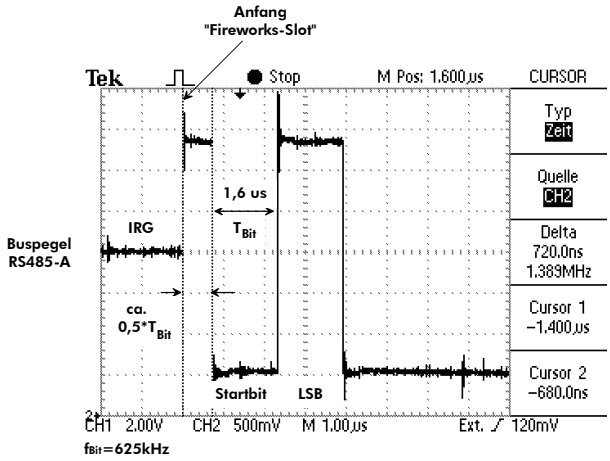


Bild 4.21: Automatischer Phasenabgleich im C16x-„Master“ bei 625 kBit/s

Den Erfolg des vorgestellten Konzepts zeigt das Oszillogramm in Bild 4.21. In diesem Beispiel sind die ersten Bits eines periodischen „Fireworks-Frames“ zu sehen. Sie stammen von einem C16x-„Master“ mit RS485-„Physical-Layer“ und einer max. Baudrate von 625 kBit/s (Grenze des µCs); weitere Details befinden sich in Kapitel 5 und im Quellcode auf der CD im Anhang. Vergleicht man das Oszillogramm in Bild 4.21 mit dem aus Bild 4.16, so stellt man keinen verwischten „Graubereich“ fest. Diese Absenz zeugt nicht nur von der „Jitter“-Freiheit der „Fireworks-Frames“, sondern von der „Jitter“-Freiheit aller „Master-Frames“. Mit einem Software- oder „resetable“ UART – sofern man die

„Reset“-Funktion benutzt – ist das in den meisten Fällen nicht möglich (siehe Abschnitt 4.1, insbesondere Abschnitt 4.1.2).

Um Fehlinterpretationen seitens der „Slaves“ von Haus aus zu vermeiden, sollten die Test-„Frames“ nicht auf dem Bus erscheinen. Hierfür kann man die Diagnosefunktion „Loop Back“ verwenden, die viele UARTs unterstützen. Sie stellt eine interne Verbindung zwischen Sender und Empfänger her, ohne dass die Testbits nach außen hin sichtbar werden. Dazu ist es erforderlich, den TxD-Pin des µCs kurzfristig als Eingang zu konfigurieren. Allerdings ist in dieser Zeit der TxD-Pin hochohmig und unter Unterständen sein Pegel nicht definiert. Es kann deshalb notwendig sein, den hochohmigen TxD-Pin mit einem externen Widerstand auf logisch „1“ (Stoppbit) zu ziehen, damit der Buspegel richtig und vor allem ruhig ist. Die Maßnahmen hängen natürlich stark vom verwendeten „Physical-Layer“ ab. Mit RS485 sind sie z.B. nicht erforderlich. Hier sorgt der einstellbare hochohmige Zustand des „Transceivers“ für klare Verhältnisse. Bei einem CAN- oder ISO 9141-„Physical-Layer“ sind diese oder entsprechend andere Maßnahmen notwendig. Falls der UART kein „Loop Back“-Funktion besitzt, ist der Aufwand ein bisschen größer (z.B. Gatterbaustein). Allerdings trifft das eher selten zu, weil der „Master“ in der Regel mit keinem „low-cost“ µC ausgestattet ist und nur „low-cost“ µCs an dieser Stelle sparen. In Ausnahmefällen dürfen die Test-„Frames“ auch auf dem Bus erscheinen. Dann muss das Test-„Frame“ aber eine max. Codedistanz zu den „Fireworks“-Codes (siehe Tabelle 3.3) aufweisen. Es eignet sich z.B. ein „Frame“ mit 0x55 als Datum und ungerader Parität.

4.2.5 „Slave“-Rundenstart

Prinzip

Aufgrund des ereignisorientierten „Master/Slave“-Anteils in „Basic-TTP/A“ und der Akkumulation der Uhrenfehler, muss ein „Slave“ den „Interleave“-Algorithmus nach jedem „Fireworks-Frame“ neu starten. Je nach Implementierung und Baudrate, können diese regelmäßigen Rundenstarts zeitkritisch sein oder nicht. Sie sind es, wenn ein „Slave“ die Fähigkeit besitzen soll im ersten Daten-„Slot“ einer oder mehrerer Runden teilnehmen zu können und die Baudrate hoch ist. In diesem „Worst-Case“ steht ihnen als Rechenzeit ungefähr $2 \cdot T_{Bit,\,M}$ zur Verfügung[106], was bei Baudraten von einigen 100 kBit/s nur wenigen µs entspricht. Die Misere entsteht durch die mäßige Rechenleistung von „low-cost“ µCs (5..10 MIPS) und der Menge an Aufgaben bei einem Rundenstart. Letztere können leider nicht per „Interleaving“ vorbereitet werden, da die Voraussetzung der Planbarkeit, so wie sie in den Runden durch das TDMA-Schema gegeben ist, fehlt. Bild 4.22 soll diesen Sachverhalt verdeutlichen. Es zeigt den prinzipiellen Ablauf des Rundenstarts in einem „Slave“. Im Übrigen wird davon ausgegangen, dass die Uhrenfehler des „Slaves“ innerhalb der Toleranzgrenzen liegen. Die erforderlichen Maßnahmen wurden in Abschnitt 4.1 beschrieben. Auf ihren Einbau in das Programmkonzept gehen die Unterpunkte im Anschluss ein.

Der Rundenstart muss durch den „Receive-Interrupt“ im „Fireworks-Slot“ initiiert werden. Folglich liegt es nahe, die Aufgaben in eine separate „Task“ bzw. ISR zu packen und mit dem „Receive-Interrupt“ zu verbinden. Auf diese Weise wird die so genannte „Round-Start-Task“ automatisch und ohne großen Aufwand durch jedes „Fireworks-Frame“ aktiviert; bezüglich der Priorität gilt das Gleiche wie für die „Slot-Task“. Nach dem Eintritt in die „Round-Start-Task“ erfolgt eine Prüfung des „Fireworks-Frames“ gemäß Abschnitt 3.2.8. Aus Geschwindigkeitsgründen wird die, in dem gleichen Abschnitt beschriebene, tabellarische Speicherung der sechszehn „Fireworks“-Codes empfohlen. Wenn der „Fireworks“-Code gültig ist, geht die „Round-Start-Task“ von einer fehlerfreien Übertragung aus und

[106] „Fireworks-Frame“ haben immer die Ideallänge und „jittern“ nicht.

extrahiert die Rundennummer. Danach sieht sie in der RODL nach, ob der „Slave“ in dieser Runde teilnimmt. Im Positivfall werden die „Schedule“-Informationen geladen und der Start des „Interleave“-Algorithmuses nach Abschnitt 4.2.3 vorbereitet. Anschließend erfolgt die Programmierung des „Slot-Timers“ unter Beachtung seiner Lagefehlerkorrektur. Das Prinzip ist links neben dem PAP in Bild 4.22 angedeutet; die Details befinden sich in Abschnitt 4.1.2. Im letzten Schritt wird der „Receive-Interrupt“ gesperrt, um eine Aktivierung durch die Daten-„Frames“ zu verhindern. Alles Weitere übernimmt die „Slot-Task“ bzw. der darin enthaltene „Interleave“-Algorithmus. Am Ende einer Runde, in der IRG, schaltet die „Slot-Task“ den „Receive-Interrupt“ wieder frei (Block „prepare next round“ in Bild 4.15), damit die „Round-Start-Task“ durch das nächste „Fireworks-Frame“ angestoßen wird und der Ablauf aufs Neue beginnt[107].

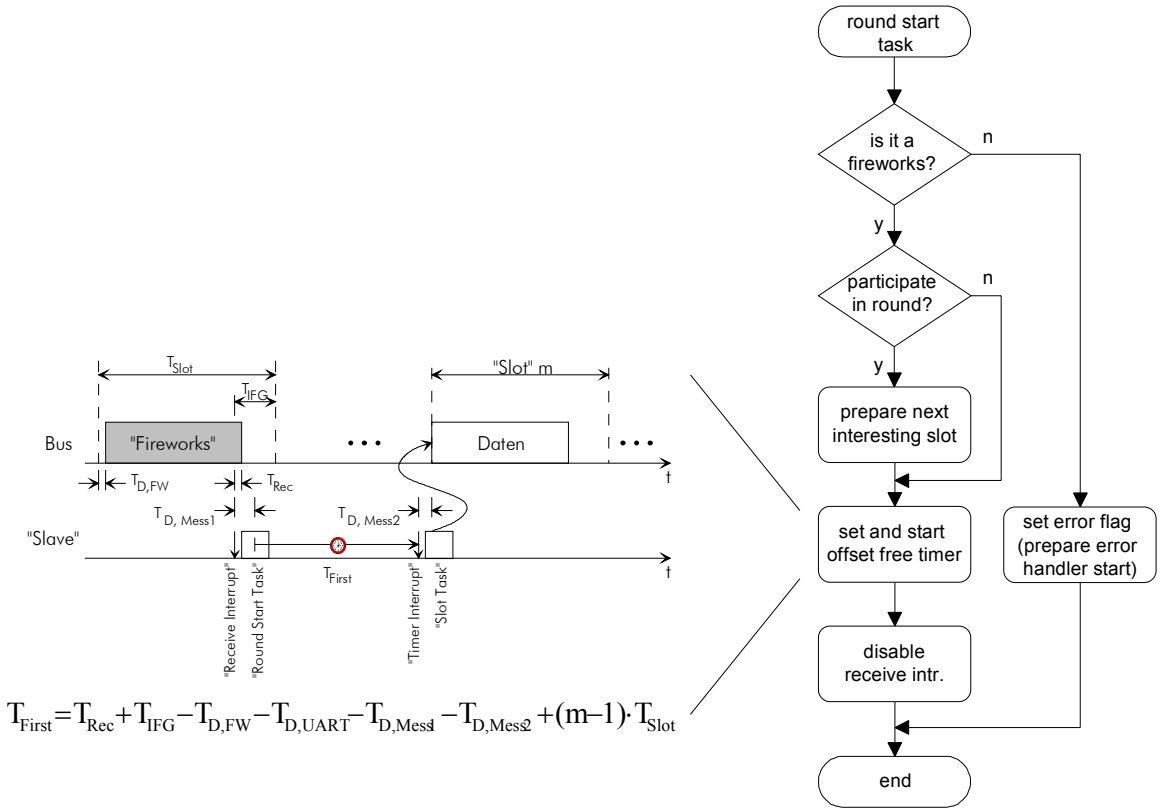


Bild 4.22: Prinzip des Rundenstarts eines „Slaves“

Ein ungültiger „Fireworks“-Code führt zu einer Fehlermeldung, die nach Beendigung der „Round-Start-Task“ von der Applikation oder einem „Error-Handler“ bearbeitet wird. Auf diese Weise kann ein „Slave“ z.B. einen „Fail-Safe“-Zustand einnehmen, wenn der Fehler häufig auftritt – das „Booten“ ist eine Ausnahme. Von einer Vorwärtskorrektur wird abgeraten, weil sie byzantinische Fehler[108] und damit eine Fehlerfortpflanzung, auch über mehrere Runde, begünstigt. Nach dem Erzeugen der Fehlermeldung wird die „Round-Start-Task“ abgebrochen, ohne den „Receive-Interrupts“ abzuschalten. Ansonsten würde sich ein „Slave“ nach dem ersten ungültigen „Fireworks“-Code schachmatt setzen.

[107] Ein „Slave“ mit „Tristate-Physical-Layer“ und einem µC ohne „Transmit-Interrupt“ benötigt den „Receive-Interrupt“ auch zum Abschalten des Bustreibers. Hier darf der „Receive-Interrupt“ nicht gesperrt werden. Stattdessen ist ein „Flag“ und ein kleiner „Interrupt-Handler“ notwendig.

[108] Ein byzantinischer Fehler ist ein Fehler, der von den Teilnehmern unterschiedlich wahrgenommen wird.

Es läuft ja weder die „Slot-Task“ an, noch kann eine Freischaltung durch die Applikation vorausgesetzt werden. Die Einbuße der „Hot-Plug-In“-Fähigkeit wäre z.B. eine Folge. Schließlich ist die Wahrscheinlichkeit sehr hoch, dass ein „Slave“ nach dem Anstecken an ein laufendes „Basic-TTP/A“-System irgendeine fallende Signalflanke in einer Runde als Startbitflanke interpretiert, wodurch zwangsweise „Fireworks“-Fehler entstehen.

Bei einem gültigen „Fireworks“-Code und der Nichtteilnahme an einer Runde wird der Bearbeitungsschritt für die Vorbereitung eines „Slots“ übersprungen. Alles weitere hängt von der Implementierungsart ab. Im einfachsten Fall programmiert die „Round-Start-Task“ den „Slot-Timer“ auf das Rundenende, wodurch die Applikation bis zur IRG ungestört rechnet. Dort erfolgt die programmierte Unterbrechung durch die „Slot-Task“, um den „Receive-Interrupt“ wieder freizuschalten und, um z.B. Korrekturrechnungen für die Uhren durchzuführen. Legt man hingegen auf Sicherheit oder Universalität Wert, sollte die „Slot-Task“ in jedem „Slot“ aufgerufen werden. Dadurch kann ein „Slave“ das Busgeschehen verfolgen und zusätzliche Fehler erkennen. Außerdem wird die dynamische Umprogrammierung der RODL erleichtert und deren Größe, bei einem „Slave“, der viele „Slots“ belegt, reduziert. Über dieses Thema wurde insbesondere in Kapitel 3 berichtet, weshalb die Ausführungen hierüber genügen sollen. Viel wichtiger ist die Erkenntnis, dass die Details und das Zusammenspiel der „Round-Start-“ und „Slot-Task“ unterschiedlich sein können und von den Anforderungen bzw. der Implementierungsart abhängen.

Beschleunigung

Eine Variante, mit der die Rechenzeit unter den eingangs erwähnten „Worst-Case“-Bedingungen besser nutzbar ist, wird nachfolgend vorgestellt. Sie ermöglichte den Betrieb von „Basic-TTP/A“ für die universelle C16x-Portierung aus Kapitel 5 bis 625 kBit/s. Bei dieser Baudrate dauert ein Bit lediglich 1,6 µs. Folglich muss ein „Slave“, der im ersten Daten-„Slot“ teilnehmen soll, innerhalb von etwa 3 µs ($\approx T_{IRG}$) alle beschriebenen Aufgaben erledigen. Leider gehört dazu auch der Wechsel zwischen der „Round-Start-Task“ und der „Slot-Task“. Durch eine aufwandarme Realisierung der „Tasks“ als ISRs, gewinnt man Rechenzeit für die eigentlichen Aufgaben. Wegen der mäßigen Rechenleistung von „low-cost“ µCs, macht sich der „Task“-Wechsel, unter diesen Umständen, jedoch immer bemerkbar. Er beträgt beim ca. 10 MIPS starken C16x-µC, der sich bezüglich seiner Rechenleistung eher im oberen Bereich der „low-cost“ µCs befindet, und einer bedachten Programmierung ungefähr 1 µs (siehe Abschnitt 4.2.2); das entspricht in etwa zehn Maschinenbefehlen und ist für einen „Interrupt-Exit“ und einen „Interrupt-Entry“ relativ wenig. Für die Aufgaben der „Round-Start-Task“ und dem ersten Teil der „Slot-Task“ verblieben in diesem Beispiel also lediglich 2 µs Rechenzeit bzw. ca. 20 Maschinenbefehle[109]. Aus den praktischen Erfahrungen kann berichtet werden, dass die angegebene max. Baudrate der universellen C16x-Portierung damit nicht erreichbar ist.

Deshalb ist eine Modifikation der „Round-Start-Task“ gemäß Bild 4.23 ratsam, wenn die Baudrate hoch ist und ein „Slave“ den ersten Daten-„Slot“ belegt. Prinzipiell ändert sich, gegenüber dem Ablauf in Bild 4.22, nicht viel. Anstatt die „Slot-Task“ für die Bearbeitung des ersten Daten-„Slots“ zu aktivieren, übernimmt die „Round-Start-Task“ die Aufgabe selbst. Wie gehabt, bereitet sie den ersten Daten-„Slot“ vor, programmiert und startet den „Slot-Timer“. Dann wartet sie jedoch auf das Signal vom „Slot-Timer“, um den „Slot“ mit min. Verzögerung auszuführen. Anschließend erfolgt die Einleitung der Übergabe an die „Slot-Task“. Diese geschieht beim nächsten „Slot“, den der „Slave“ belegt. Alles weitere läuft in gewohnter Weise ab. Der offensichtliche Vorteil dieser Beschleunigungsmethode ist der eingesparte „Task“-Wechsel an der zeitkritischsten Stelle für einen „Slave“. Zur Entspannung trägt aber auch die wesentlich geringere Zeitvarianz bei. Da die „Round-Start-Task“ die

[109] Für einen einfachen C-Befehl kann man ungefähr drei bis vier Maschinenbefehle ansetzen.

CPU exklusiv belegt und gezielt auf das „Slot-Timer“-Ereignis wartet, können keine Verzögerungen durch „Critical-Sections“ o.Ä. auftreten. Unter Berücksichtigung des sehr geringen Gangfehlers im ersten Daten-„Slot“, werden ungefähr $0{,}7 \cdot T_{Bit,\,M}$ Pufferzeit nicht benötigt. Die „Round-Start-Task“ darf diesen Freiraum, der bei 625 kBit/s mehr als 1 µs beträgt, zum Rechnen nutzen, ohne eine Realzeitanforderung des Protokolls zu verletzen. Eindrücke über die Rechenzeitverhältnisse in einem C16x-„Slave“ bei 125 und 625 kBit/s vermitteln die Oszillogramme in Bild 4.26 (S. 123) und Bild 5.6 (S. 143).

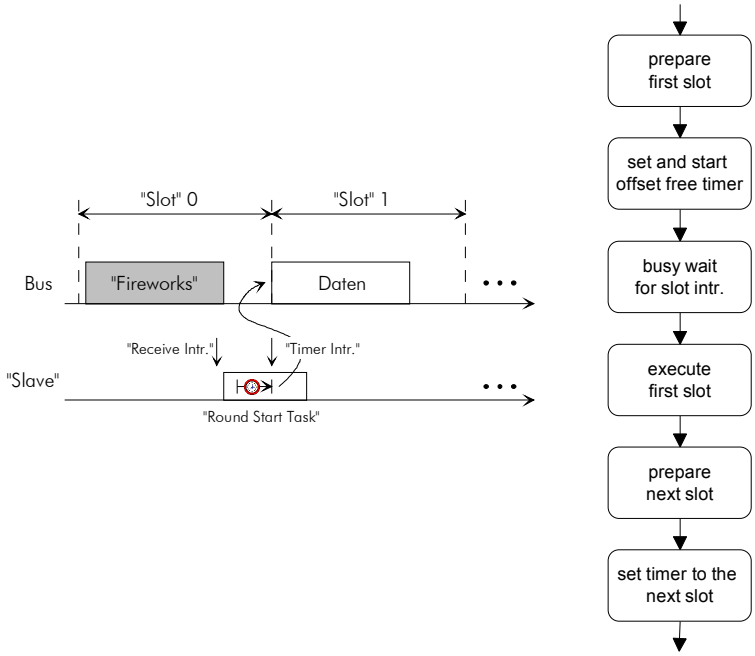


Bild 4.23: Prinzip der Rundenstartbeschleunigung bei „Worst-Case“-Bedingungen

Die Methode hat jedoch auch Nachteile. Sie ist eine Speziallösung, weil sie viel Rechenzeit beim Warten („Busy-Waiting“) vergeudet, falls die geschilderten Voraussetzungen (hohe Baudrate, erster Daten-„Slot“ belegt) nicht zutreffen. Deshalb sollte man sie nur anwenden, wenn der ursprüngliche Algorithmus tatsächlich zu langsam ist. Außerdem wird etwas mehr Speicher benötigt, da die „Round-Start-Task“ temporär die Aufgabe der „Slot-Task“ übernimmt, wodurch ihr Umfang und ihre Komplexität entsprechend wächst. Der Speichermehrbedarf ist jedoch nicht so groß (≈ 50..100 Bytes Code), als dass sich die Methode nicht für eine Implementierung auf „low-cost“ µCs eignet (siehe Abschnitt 5.2.6).

Für eine Speicherplatzoptimierung kann man die „Round-Start-Task“ selbstverständlich in die „Slot-Task“ integrieren. Allerdings leidet die Geschwindigkeit darunter, weil dann am Anfang der „Slot-Task“ ein „Interrupt-Handler“, aufgrund der Fallunterscheidung zwischen „Receive-“ und „Slot-Timer-Interrupt“, notwendig ist. Von einer alleinigen Steuerung durch den „Slot-Timer-Interrupt“ muss abgeraten werden. Hier würde der „Slave“ am „Busy-Waiting“-Punkt die CPU blockieren, sobald der „Master“ ausfällt. Dadurch wäre ein „Slave“ nicht in der Lage, eigenständig einen „Fail-Safe“-Zustand o.Ä. einzunehmen. Aus dem gleichen Grund ist auch eine vorzeitige Aktivierung des Rundenstarts mit „Busy-Waiting“ auf den „Fireworks-Receive-Interrupt“, zur Verbesserung der Reaktionszeit, gefährlich und deshalb nicht empfehlenswert.

4.2.6 „Slave“-Uhrenfehler

Über die möglichen Uhrenfehler in einem „Slave“, die damit verbundenen Probleme und entsprechende Lösungen, wurde ausführlich in Abschnitt 4.1 berichtet. An dieser Stelle werden nun Vorschläge für die Integration der obligatorischen „Slot-Timer“-Lagefehlerkorrektur (siehe Abschnitt 4.1.2), der optionalen „Slot-Timer“-Gangfehlerkorrektur (siehe Abschnitt 4.1.3) und der, mit Letzterem verbundenen, optionalen UART-Synchronisation (siehe Abschnitt 4.1.4) in das Programmkonzept gemacht.

„Slot-Timer“-Lagefehlerkorrektur

Voraussetzung für die „Slot-Timer“-Lagefehlerkorrektur ist die Kenntnis der systematischen, konstanten Zeitversätze, gegenüber den Idealbedingungen. Nach Formel 4.7 in Abschnitt 4.1.2, bzw. Bild 4.22 in Abschnitt 4.2.5 müssen dazu die Latenz- und Ausführungszeiten der „Round-Start-Task“ und „Slot-Task“ vermessen werden. Das Prinzip ähnelt dem Phasenabgleich im „Master“ aus Abschnitt 4.2.4 und ist in Bild 4.24 zu sehen. Es werden parallele, ebenso lange Programmpfade bis zu den entscheidenden Punkten (grau unterlegt) in der „Round-Start-Task“ und „Slot-Task“ aufgebaut. Diese Programmpfade werden beim „Booten“ einmal durchlaufen und die entsprechenden Zeiten z.B. mit dem „Slot-Timer“ gestoppt[110] – in diesem Zustand ist der „Slot-Timer“ noch frei. Anschließend erfolgt die Berechnung der ersten oder aller „Slot-Timer“-Werte, um Rechenzeit während des Betriebs zu sparen. Je nach Implementierung und Ressourcen, kann es hier große Unterschiede geben. Es ist sicherlich sinnvoll alle „Slot-Timer“-Werte a priori zu berechnen, wenn ein „Slave“ nur an ein paar „Slots“ teilnimmt. Belegt er hingegen viele „Slots“, kann man Speicherplatz sparen, wenn der „Slave“ alle oder die zumindest die hinteren „Slot-Timer“-Werte in einer Runde über die „Slot“-Nummern ad hoc ermittelt. Denn im Vergleich zu einem 16 Bit langen „Slot-Timer“-Wert, benötigt eine „Slot“-Nummer nur 8 Bit.

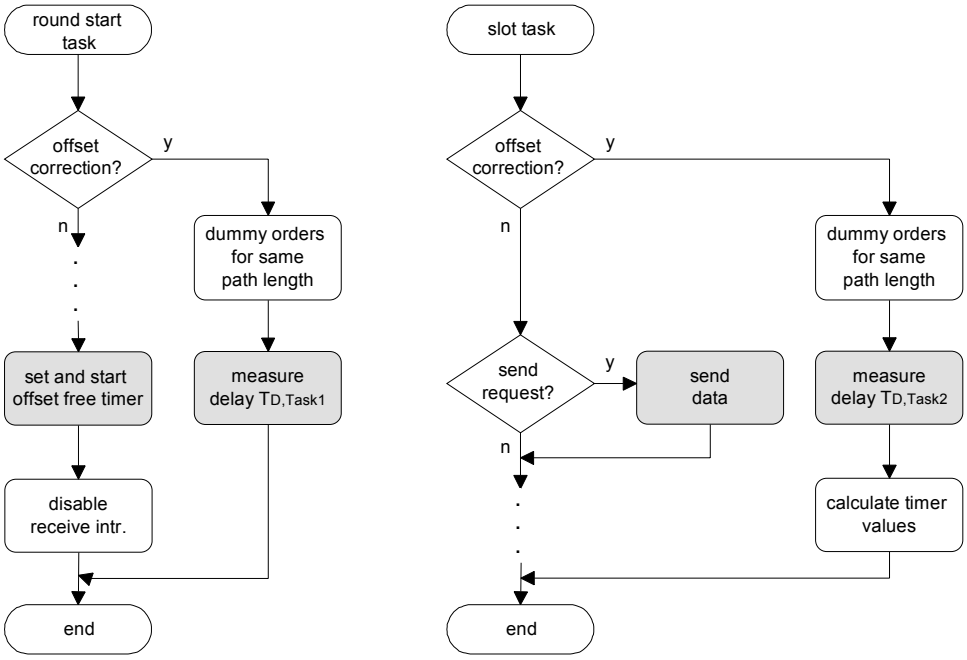


Bild 4.24: Erweiterungen der „Slave-Tasks“ um die Lagefehlerkorrektur

Eine gewisse Implementierungsfreiheit besteht auch bezüglich der Messungen. Diese müssen nicht in einem parallelen Programmpfad durchgeführt werden. Man kann das genauso im eigentlichen Programmpfad machen. Das spart sogar ein paar Byte Codespeicher, weil keine bzw. weniger „Dummy"-Befehle nötig sind. Allerdings werden die Messvorbereitungen, aufgrund der größeren Anzahl an Programmverzweigungen, aufwendiger und der Quellcode unübersichtlicher. Vorabmessungen, z.B. durch Auszählen der Maschinenbefehle und Nachschlagen der Latenzzeiten, sind ebenfalls möglich. Damit lässt sich noch mehr Speicher einsparen. Wegen so genannter „Magic-Numbers", leiden die Lesbarkeit, Portierbarkeit und vor allem die Handhabbarkeit der Implementierung stark darunter. Aus diesen Gründen bevorzugt der Autor die etwas aufwändigere, beschriebene Messtechnik. Ferner sind entsprechende Modifikationen notwendig, sollte man von der Beschleunigungsmethode oder dem Einbau der „Round-Start-Task" in die „Slot-Task" Gebrauch machen (siehe Abschnitt 4.2.5). Die Unterschiede sind jedoch gering, weil das Prinzip gleich bleibt.

Um die Bedeutung der „Slot-Timer"-Lagefehlerkorrektur zu unterstreichen, wurde ein Vergleich zwischen einem C16x-„Slave" mit und ohne Korrektur bei 125 kBit/s erstellt. Das Ergebnis ist in dem Oszillogramm in Bild 4.25 zu sehen. Man erkennt deutlich den Unterschied bei der „Frame"-Platzierung. Ohne Korrektur tritt eine zusätzliche Verzögerung von ca. 5,6 µs auf, was bei der gewählten Baudrate in etwa $0{,}7 \cdot T_{Bit}$ entspricht. Unter „Worst-Case"-Bedingungen, wenn alle Fehler max. groß und gleichgerichtet sind, würde dieser „Slave" die Realzeitbedingungen des Protokolls verletzen. Nach Abschnitt 4.1.2 muss der „Slot-Timer"-Lagefehler im „Worst-Case" nämlich kleiner als $0{,}3 \cdot T_{Bit}$ sein. Dass der C16x-„Slave" bei 125 kBit/s trotzdem funktioniert, liegt an seinem Quarzoszillator. Er besitzt so gut wie keinen „Slot-Timer"-Gang- und Baudratenfehler, wodurch der „Worst-Case" nicht gegeben ist. Ohne „Slot-Timer"-Lagefehlerkorrektur wäre der C16x-„Slave" ungefähr bis 200 kBit/s betreibbar. Dann würde sein „Slot-Timer"-Lagefehler und der nicht zu vergessende UART-„Jitter" die gesamte IFG aufgebraucht haben. Schließlich verschlechtern sich die Verhältnisse mit zunehmender Baudrate, weil die Verzögerung konstant ist, währenddessen die Rechenzeit abnimmt. Gegenüber den erreichten 625 kBit/s mit „Slot-Timer"-Lagefehlerkorrektur ist das ein Unterschied um mehr, und im „Worst-Case" um weit mehr, als den Faktor drei.

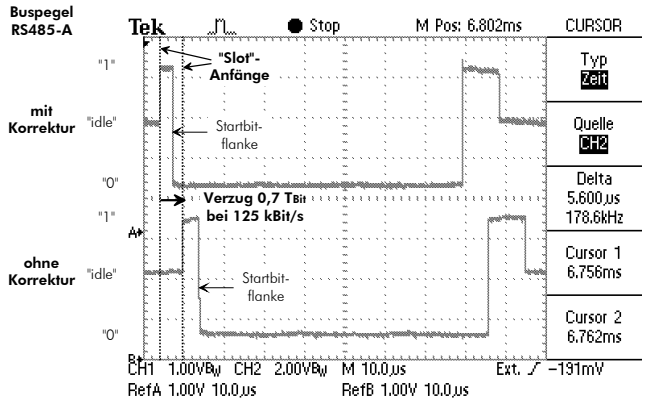


Bild 4.25: Wirkung der „Slot-Timer"-Lagefehlerkorrektur (125 kBit/s, C16x-„Slave")

Außerdem liefert der Vergleich eine Vorstellung über die Rechenarbeit beim Rundenstart. Die zusätzlichen 5,6 µs entstehen beim C16x-„Slave" hauptsächlich durch die Latenz- bzw. „Task"-Wechselzeit der „Round-Start-Task" – der C16x-„Slave" arbeitet mit der beschriebenen Beschleunigungsmethode

[110] Die zeitinvarianten bzw. nahezu zeitinvarianten Befehle in „low-cost" µCs ermöglichen diese Technik.

– und die Befehlsausführungszeiten. Wie geschildert, hat der C16x-„Slave“ nahezu keinen „Slot-Timer“-Gang- und Baudratenfehler. Des Weiteren kompensiert der Zeitgewinn T_{Rec}, aufgrund des „Receive-Interrupts“ nach $9/16 \cdot T_{Bit}$ im Stoppbit, den „Fireworks“-Verzug $T_{D, FW}$ nahezu vollständig. Folglich verbleiben als Ursachen nur die gemessene „Task“-Wechsel- und Rechenzeit (siehe Formel 4.7 in Abschnitt 4.1.2). Abzüglich der „Task“-Wechselzeit, rechnet der C16x-µC bis zum kritischen Punkt des Sendens etwa 5 µs. In dieser Zeit führt er, mit seinen max. 10 MIPS Rechenleistung, ca. 50 Maschinenbefehle bzw. 10..15 C-Befehle aus. Vergleicht man darüber hinaus diese Rechenzeit mit der Abschätzung bei 625 kBit/s aus Abschnitt 4.2.5, so stellt man Folgendes fest: Erstens ist bei diesem µC und dieser Baudrate die Beschleunigung der „Round-Start-Task“ notwendig. Und zweitens stellt 625 kBit/s nicht nur die max. Baudrate des C16x-µCs dar, sondern auch die max. Baudrate der Implementierung mit diesem µC.

„Slot-Timer“-Gangfehlerkorrektur und UART-Synchronisation

Der letzte Punkt in dem Programmkonzept betrifft die „Slot-Timer“-Gangfehlerkorrektur und die UART-Synchronisation. Aus den Abschnitten 4.1.3 und 4.1.4 ist bekannt, dass ein „Slave“ mit RC- oder „On-Chip“-Oszillator beide Kompensationsmechanismen benötigt, ein „Slave“ mit Keramikoszillator nur über die „Slot-Timer“-Gangfehlerkorrektur verfügen muss und in einem „Slave“ mit Quarzoszillator keines von beiden erforderlich ist. Des Weiteren wurde gezeigt wie diese Mechanismen arbeiten und welche Mathematik ihnen zugrunde liegt. Aus diesem Grund werden diese Punkte hier oberflächlich behandelt und stattdessen auf ihren Einbau in das Programmkonzept eingegangen.

Die Vermessung der Zeitspanne zwischen zwei „Fireworks-Frame“ betrifft nur die „Round-Start-Task“. Sie überprüft anhand von Formel 4.11 (siehe Abschnitten 4.1.3), ob der Messzeitraum groß genug. Wenn das zutrifft, speichert sie den Wert T_{Mess} für die Prädiktion zukünftiger „Slots“ ab. Ansonsten wird mit dem alten Messwert weitergerechnet. Die Gangfehlerkorrektur des „Slot-Timers“ nach Formel 4.9 (siehe Abschnitten 4.1.3) findet entweder in „Round-Start-Task“, in der „Slot-Task“, in beiden oder in einer niederprioren „Task“ statt. Das hängt von der Baudrate, den µC-Ressourcen – insbesondere der Rechenleistung und des Speichers – und der Implementierungsart ab. Häufig belegt ein „Slave“ nur wenige „Slots“. In diesen Fällen können alle „Slot-Timer“-Werte gespeichert werden. Aufgrund der dann kurzen Reaktionszeiten, lassen sich entsprechend hohe Baudraten erzielen. Für die Aktualisierung der „Slot-Timer“-Werte kann man rechenfreie Zeiten nutzen, da die gangfehlerverursachende Oszillatordrift langsam vonstatten geht. Außerdem ist es absolut unproblematisch, wenn ein „Slave“ während der Aktualisierung einmal mit einem neuen und ein anderes Mal mit einem alten „Slot-Timer“-Wert arbeitet.

Wenn ein „Slave“ dagegen an vielen „Slots“ teilnimmt und über wenig Speicher verfügt, kann eine Ad-Hoc-Berechnung der 16 Bit „Slot-Timer“-Werte über die 8 Bit „Slot“-Nummern erforderlich sein. Die max. Baudrate fällt entsprechend niedrig aus, da die Berechnungen in der „Round-Start-Task“ und/oder „Slot-Task“ durchgeführt werden müssen und somit harte Zeitbedingungen herrschen. Etwas günstiger sieht die Sache bei aufeinanderfolgenden oder äquidistanten „Slots“ aus. Hier kann ein „Slave“ mit einem Start- und Differenzwert für den „Slot-Timer“ an mehreren „Slots“ ressourcenschonend teilhaben. Das gilt besonders für „Slaves“ mit einem µC mit „Auto-Reload-Timer“. Ein Nachteil dieser Methode ist die Fehlerfortpflanzung durch den Rundungsfehler des Differenzwerts (siehe Abschnitt 4.1.3). Aus diesem Grund dürfen damit nicht zu viele „Slots“ vorhergesagt werden. Trotzdem ist diese Methode praxisrelevant, weil damit schätzungsweise bis zu zehn aquidistante „Slots“ pro Runde vorhersagbar sind.

Ein ganzes Stück einfacher ist die Synchronisation des UARTs. Wegen der größeren Toleranz, ist die Aktualisierung des Bittakts noch weniger zeitkritisch als die „Slot-Timer“-Gangfehlerkorrektur. Au-

ßerdem sind die µC-Ressourcen sekundär, da nur ein Wert in einem reservierten Register, gemäß Formel 4.12 (siehe Abschnitt 4.1.4), korrigiert werden muss. Das einzig Beachtenswerte ist der Zeitpunkt der Aktualisierung. Oftmals stört eine Umprogrammierung der Baudrate den Sende- und/oder Empfangszyklus, weil viele UARTs dabei ihre Steuerungen und/oder Teiler zurücksetzen. Deshalb sollte die Baudrate nur korrigiert werden, wenn der UART nicht arbeitet. Hierfür eignen sich Runden, an denen ein „Slave“ nicht teilnimmt, längere Pause in einer Runde oder die IRG. Letztere hat den Vorteil der Applikationsunabhängigkeit und ist zudem in allen vorgestellten Implementierungsvarianten einsetzbar. Des Weiteren hat ein „Slave“ nach dem Rundenende wenig zu tun und einen großen Spielraum, wie Bild 4.26 zeigt.

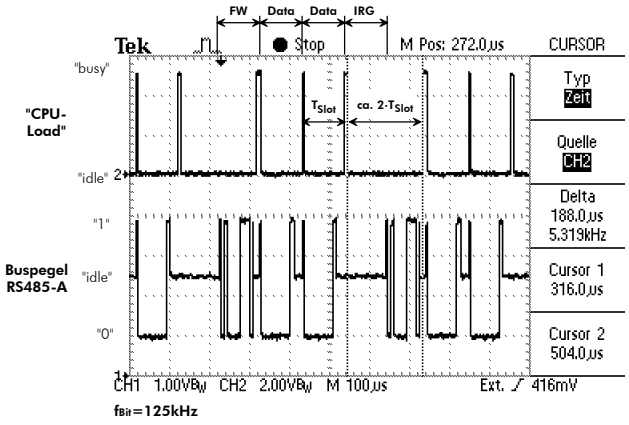


Bild 4.26: CPU-Belastung beim C16x-„Slave“

In dem Beispiel wurde die Rechenzeit von „Basic-TTP/A“ (oben), mit der C16x-Implementierung aus Kapitel 5, bei 125 kBit/s vermessen. Man erkennt deutlich die größeren Spielräume an den Rundenenden – die C16x-„Slaves“ haben Quarzoszillatoren und benötigen deshalb weder eine „Slot-Timer“-Gangfehlerkorrekur, noch eine UART-Synchronisation. Sie entstehen zum einen, aufgrund der geringen Vorbereitungen für die nächste Runde und zum anderen, weil „Basic-TTP/A“ in einem „Slave“ erst nach dem „Fireworks-Frame“ wieder aktiv wird. Die freie Rechenzeit beträgt in diesem Beispiel knapp zwei Bitzeiten bzw. ca. 188 µs. In dieser Zeit wäre der C16x-µC in der Lage, etwa 1880 Maschinenbefehle auszuführen.

Es lässt sich leicht nachvollziehen, dass das für weit mehr als die UART-Synchronisation ausreicht. Aus diesem Grund liegt es nahe, die „Slot-Timer“-Gangfehlerkorrektur dort ebenfalls durchzuführen. Glücklicherweise ist das in den meisten Fällen möglich. Wie geschildert, belegt ein „Slave“ in der Regel nur ein paar „Slots“ oder kann mit dem ressourcensparenden „Auto-Reload“-Mechanismus arbeiten. Für diese „Slaves“ werden die in Bild 4.27 gezeigten Erweiterungen der „Round-Start-Task“ und „Slot-Task“ empfohlen. Die zeitkritische „Round-Start-Task“ kümmert sich lediglich um die Vermessung der „Fireworks-Frames“ und die „Slot-Task“ um den Rest. Auf diese Weise wird keine zusätzliche „Task“ für die „Slot-Timer“-Gangfehlerkorrektur und die UART-Synchronisation benötigt, was den µC-Ressourcen zugute kommt. Außerdem werden durch die geordnete, serialisierte Bearbeitung von vornherein Ressourcenkonflikte unterbunden. Ohne zusätzliche Maßnahmen bestünde dafür hauptsächlich bei der Aktualisierung der 16 Bit „Slot-Timer“-Werte auf 8 Bit µCs Gefahr.

Die zusätzlichen Aufgaben der „Slot-Task“ konzentrieren sich ausschließlich auf das Rundenende. Wenn die „Round-Start-Task“ einen neuen Messwert liefert, dann berechnet die „Slot-Task“ mit dem Beginn der nächsten IRG die neuen „Slot-Timer“-Werte und die neue Baudrate. Außerdem program-

miert sie das Baudratenregister um. Auf diese Weise entsteht zwar eine Verzögerung der Aktualisierung von einer Runde, angesichts einer Messperiode von ca. 2..3 s (siehe Abschnitt 4.1.3) spielt das aber keine Rolle.

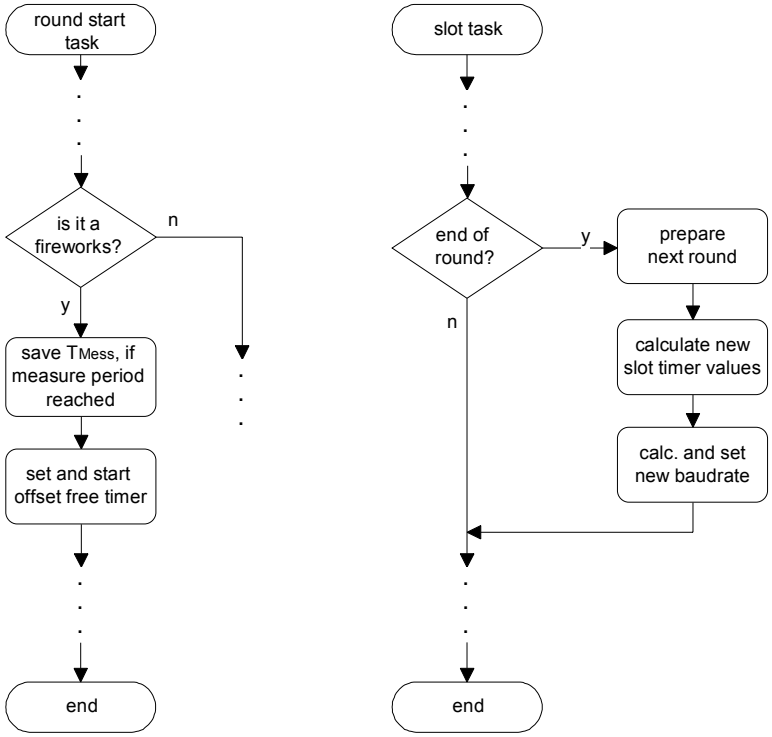


Bild 4.27: Empfohlene Erweiterungen der „Slave-Tasks“ für die Kompensation der Oszillatordrift

4.3 Bewertung

4.3.1 Zeitverhalten

Das Zeitverhalten wird durch „Basic-TTP/A“ und die Applikationen festgelegt. Für die zeitliche Platzierung der Runden und „Slots“ zeichnet „Basic-TTP/A“ verantwortlich. Die Applikationen hingegen definieren die Baudrate und die „Schedules“. Außerdem übernimmt die „Master“-Applikation die Rundensteuerung. Sie kann zyklisch, ereignisgesteuert oder, durch eine übergeordnete Instanz, ferngesteuert sein. Wegen diesen applikationsspezifischen Anteilen, ist es unmöglich eine generelle Aussage über das Zeitverhalten zu machen. Für „Basic-TTP/A“ alleine ist das aber durchaus machbar. Allgemein gilt, dass „Basic-TTP/A“ Daten zeitlich deterministisch (in den „Slots“) und mit geringem „Jitter“ überträgt. Insbesondere diese Eigenschaft hebt „Basic-TTP/A“ von anderen Sensor/Aktorbussen ab, wie Kapitel 1 und 2 gezeigt haben.

Integrales Zeitverhalten

Das integrale, also langfristige Zeitverhalten von „Basic-TTP/A“ wird durch den „Master“ festgelegt. Er platziert, durch das Versenden der „Fireworks-Frames“, die Runden, an denen sich die „Slaves“ orientieren und synchronisieren. Folglich bestimmt die Genauigkeit des „Master“-Oszillators das in-

tegrale Zeitverhalten eines „Basic-TTP/A“-Netzes. Aus den Abschnitten 3.1 und 4.1 ist bekannt, dass der „Master“ stets über einen genauen Oszillator verfügt. Typischerweise ist das ein Standardquarzoszillator, der aus Kostengründen nicht mit einem Trimmkondensator feinjustiert ist. Ein Standardquarzoszillator wiederum weist laut Abschnitt 8.4 einen Frequenzfehler auf, der sich aus einem Abgleichfehler von ca. $\pm 50 \cdot 10^{-6}$, einem Temperaturfehler von ca. $\pm 30 \cdot 10^{-6}$ ($\upsilon \approx$ -20..70°C) und einem Alterungsfehler von ca. $\pm 5 \cdot 10^{-6}$ pro Jahr zusammensetzt. Für die zeitliche Änderung des Frequenzfehlers ist in erster Linie der temperaturbedingte Anteil verantwortlich. Wegen der großen thermischen Zeitkonstanten, liegt die Änderungsgeschwindigkeit bei äußeren Temperaturschwankungen im oberen s-Bereich. Damit qualifiziert sich ein „Basic-TTP/A“-Netz als Zeitreferenz für die Steuerung sehr vieler technischer Prozesse.

Differentielles Zeitverhalten

Mit dem differentiellen Zeitverhalten von „Basic-TTP/A“ ist das Zeitverhalten in den Runden gemeint. Das sind im Speziellen die Platzierung der „Slots“ und „Frames“. In Abschnitt 4.1, insbesondere aber in Abschnitt 4.1.1 wurde gezeigt, dass sich der Prädiktionsfehler der „Slots“, gegenüber dem „Master“, innerhalb eines Toleranzbandes mit den Grenzen $\pm 152 \cdot 10^{-6}$ bewegen darf. Wesentlich ungenauer ist hingegen die Platzierung der „Frames“, mit Ausnahme der „Fireworks-Frames“ bzw. „Master-Frames“. Vor allem wegen des UART-„Jitters“ (siehe Abschnitt 4.2.4) befinden sich die Daten-„Frames“ irgendwo in den „Slots“. Wenn die Synchronisation zwischen „Basic-TTP/A“ und einer Applikation, so wie in Abschnitt 4.2.3 beschrieben, durch die „Slot-Task“ gesteuert wird, hat der Lagefehler der „Frames“ keine Bedeutung. Unter dieser vorteilhaften und sinnvollen Annahme, kommt das differentielle Zeitverhalten, hinsichtlich der Genauigkeit, in die Nähe des integralen Zeitverhaltens. Dieser geringe Zeitfehler ermöglicht, mit entsprechenden Applikationen, in verteilten Anwendungen nahezu gleichzeitige Aktionen (z.B. Notaus). Aufgrund des deterministischen Lang- und Kurzzeitverhaltens, erfüllt „Basic-TTP/A“ eine wichtige Voraussetzung für die Steuerung anspruchsvoller und realzeitkritischer technischer Prozesse.

Ein kleines Zahlenbeispiel soll diese herausragende Eigenschaft verdeutlichen: Es wird eine schnelle, verteilte, zeitdiskrete Regelung mit einer Sollabtastzeit von 1 ms angenommen. Aufgrund der Frequenztoleranz des Oszillators im „Master“, die je nach Temperatur bis zu ±80 ppm betragen kann, liegt die eigentliche, mittlere Abtastzeit in einem Bereich von 0,99992..1,00008 ms. Sie darf als quasikonstant angesehen werden, da sich die Oszillatorfrequenz des „Masters“ bei Temperaturschwankungen sehr langsam ändert. Für die Abweichungen um diesen Mittelwert ist das differentielle Zeitverhalten verantwortlich. Gemäß den obigen Ausführungen, muss hier ein Fehler von ±152 ppm berücksichtigt werden. Er wirkt sich jedoch anders aus als der Fehler des integralen Zeitverhaltens. Bezug für den Fehler des differentiellen Zeitverhaltens ist die Dauer nach dem „Fireworks-Slot“ bis zum entsprechenden Daten-„Slot“. Unter der weiteren Annahme von 0,5 ms (halbe Abtastzeit) für diese Dauer, folgt als Schwankungsbreite ±76 ns. Daraus resultiert ein theoretischer, relativer Gleichzeitigkeitsfehler $\Delta T_{Ab}/T_{Ab,\,Soll} = \pm 76 \cdot 10^{-6}$. Unter Berücksichtigung von unterschiedlichen Signallaufzeiten und verschiedenen Reaktionszeiten (z.B. durch die Applikation), kann dieser Wert sicherlich um den Faktor 10 und mehr größer ausfallen. Aber selbst dann ist das zeitliche Verhalten, einer auf „Basic-TTP/A“ beruhenden Steuerung, immer noch gut und vor allem berechenbar.

Kleinste Abtastzeit

Der letzte Punkt zum Thema Zeitverhalten geht auf die kleinste, äquidistante Abtastzeit ein. Sie wird in „Basic-TTP/A“ durch das direkte Aufeinanderfolgen von IRG und „Fireworks-Slot“ auf $3 \cdot T_{Slot,\,M}$ begrenzt, und nicht etwa durch die Rundenlänge. Bild 4.28 zeigt hierzu ein Beispiel mit einem simplen „Schedule“. Man erkennt daraus zwei notwendige Bedingungen für ein 8 Bit Abtastdatum: Es muss

der erste und letzte Daten-„Slot“ in den Runden durch das Abtastdatum belegt sein. Außerdem müssen die Runden 2 + 3·(n – 1) „Slots“ enthalten. Dabei ist n die Anzahl der Abtastungen in einer Runde, die natürlich von Runde zu Runde unterschiedlich sein darf. Für ein 16 Bit oder noch größeres Datum verhält sich der Sachverhalt analog. Die hinreichenden Bedingungen sind angemessene „Schedules“ und eine passende Rundensteuerung. Mit der C16x-Implementierung lässt sich auf diese Weise die bemerkenswert kurze Abtastperiode von 62,4 µs (≈ 16 kHz) realisieren.

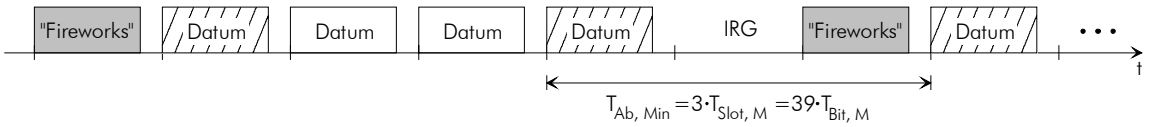


Bild 4.28: Kleinste Abtastzeit mit „Basic-TTP/A“

4.3.2 Übertragungseffizienz

Dem Protokoll aus Abschnitt 3.2 zufolge, überträgt „Basic-TTP/A“ in den 13 Bit langen „Slots“ je 8 Datenbits. Hieraus folgt eine „Slot“-Effizienz von 8 Bits/13 Bits ≈ 0,615. Wegen der „Fireworks-Slots“ und der IRGs, bildet dieser Wert eine Obergrenze für die Übertragungseffizienz von „Basic-TTP/A“. Aus Vereinfachungsgründen werden vorerst gleich lange Runden, mit jeweils s Daten-„Slots“ (s = 0..254), für die Berechnung der Übertragungseffizienz angenommen. Zuzüglich des „Fireworks-Slots“ und der IRG, dauert eine Runde $(s + 2) \cdot 13 \cdot T_{Bit,\,M}$. Von dieser Zeit belegen die Nutzdaten $s \cdot 8 \cdot T_{Bit,\,M}$. Daraus entsteht folgende Formel für die Übertragungseffizienz bei gleich langen Runden, bzw. für die Rundeneffizienz allgemein:

$$\eta_{Round} = \frac{8 \cdot s}{13 \cdot (s + 2)} \approx 0{,}615 \frac{s}{s + 2} \qquad \text{mit } s = 0..254$$

Formel 4.14: Rundeneffizienz von „Basic-TTP/A“

Für eine Diskussion ist die Funktion in Bild 4.29 graphisch dargestellt. Wegen des geringen Protokoll-„Overheads“, steigt η_{Round} sehr rasch an und erreicht bereits für s = 9 einen Wert etwas über 0,5 (ungefähr 82 % vom Endwert). Das beweist, dass „Basic-TTP/A“ mit kleinen Datenmengen, die für Sensor/Aktoranwendungen typisch sind, hocheffizient arbeitet. Andere Sensor/Aktorbusse oder vergleichbare Bussysteme sind weit weniger effizient (siehe Abschnitt 2.3 oder Tabelle 4.1 auf S. 130). Folglich kann man mit „Basic-TTP/A“ bei viel niedrigeren Baudraten dieselben Nettodatenraten erzielen. Das hat zweierlei Vorteile: Zum einen weist „Basic-TTP/A“ prinzipiell bessere EMV-Eigenschaften auf als vergleichbare Busse und zum anderen ist die erforderliche Rechenleistung der µCs in den Knoten geringer. Beides unterstützt den „Low-Cost“-Aspekt von „Basic-TTP/A“, indem z.B. einfachere und schirmlose Kabel und/oder schwächere µCs verwendet werden können.

Oberhalb von neun Daten-„Slots“ pro Runde, nimmt die Steigung der Kurve drastisch ab. Wie Bild 4.29 zeigt, erreicht man durch eine Verdopplung der Daten-„Slots“ von s = 9 auf s = 18 lediglich eine Effizienzsteigerung von ≈ 0,5 auf ≈ 0,55, also um etwa 10 %. Da sich dieser Trend fortsetzt, schmiegt sich die Kurve nur langsam an die „Slot“-Effizienz an. Erst beim Grenzübergang $s \to \infty$, was einem reinen TDMA-Betrieb entspräche, würde sie diese Asymptote berühren. Wegen der Längenbeschränkung $s_{max} = 254$, endet die Effizienz aber bei ca. 0,61. In der Praxis wird man davon meistens ein

Stück entfernt sein. Die Anzahl der „Slot“ korreliert nämlich mit der Anzahl der Sensoren und Aktoren. Ein Sensor/Aktor-„Cluster“ besitzt aber selten mehr als 30..50 Sensoren und Aktoren, so dass selbst bei ausschließlich 16 Bit-Werten die Rundenlängen deutlich unter s = 254 liegen werden.

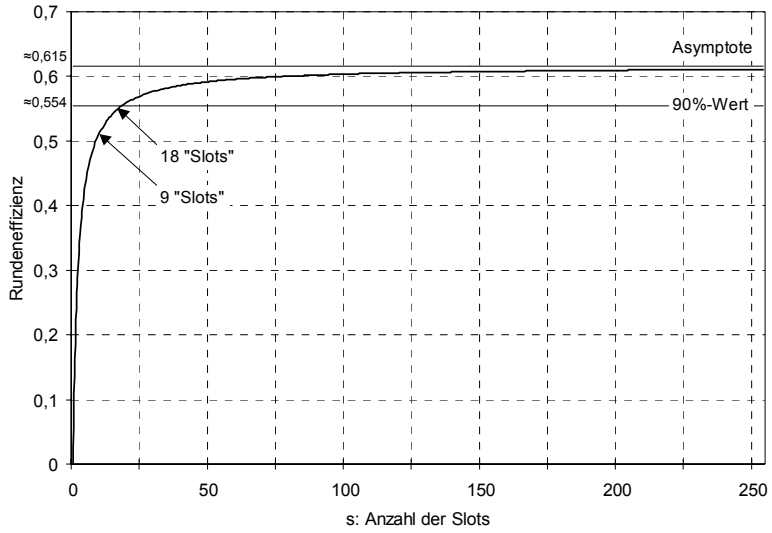


Bild 4.29: Rundeneffizienz von „Basic-TTP/A“

Kürzere Runden können auch einen Vorteil haben. Sofern der ereignisgesteuerte „Master/Slave“-Anteil in einer Anwendung wichtig ist, erhöhen kurzen Runden das Reaktionsvermögen der Rundensteuerung im „Master“. Ferner muss für einen fairen Vergleich mit den Bussystemen aus Kapitel 2 berücksichtigt werden, dass nicht jeder Sensor oder Aktor ein Daten-„Frame“ vollständig nutzt. Aus diesen Gründen ist eine typische Übertragungseffizienz von 40..50 % in einem „Basic-TTP/A“-Netz wahrscheinlich. Die exakte Übertragungseffizienz kann man im Einzelfall mittels Formel 4.15 berechnen. Sie beachtet sowohl unterschiedliche Rundenlängen als auch zyklische und sporadische Runden. In ihr stellt die Variable $\eta_{Round,\,i}$ die Rundeneffizienz der Runde i dar. Sie wird aus dem jeweiligen „Schedule“ und der Formel 4.14, unter Beachtung der „Frame“-Nutzung, ermittelt. Die Variable h_i ist hingegen die Auftrittshäufigkeit der Runde i. Für ihre Berechnung muss man die Rundensteuerung im „Master“ kennen. Allgemein gilt, dass h_i bei zyklischen Runden einfacher zu berechnen ist, als bei sporadischen Runden.

$$\eta = \sum_i \eta_{Round,\,i} \cdot h_i < 0{,}615 \qquad \begin{array}{l} \text{mit } \; i = 0..15 \;\; \text{(Rundennr.)} \\ \text{und } \; h_i = 0..1 \;\; \text{(Häufigkeit), wobei} \sum_i h_i = 1 \end{array}$$

Formel 4.15: Übertragungseffizienz von „Basic-TTP/A“

4.3.3 Verlässlichkeit

Ein wichtiger Faktor bei der Beurteilung der Verlässlichkeit eines Bussystems ist die Restfehlerwahrscheinlichkeit der Übertragung. Sie ist ein Maß für die Möglichkeit Informationen bei einer Störung nicht erkennbar zu verfälschen. Folglich sind die Informationen umso verlässlicher, je kleiner die

Restfehlerwahrscheinlichkeit ist. Diese kann für „Basic-TTP/A“ jedoch nur abgeschätzt werden, da es kein exaktes Kanalmodell gibt. Schließlich soll „Basic-TTP/A“ in sehr unterschiedlichen Störumgebungen (Kfz, Industrie, Medizin, Haus...) eingesetzt werden und mit verschiedenen „Physical-Layern“ (RS485, CAN, ISO 9141...) arbeiten können. Deshalb werden für die Restfehlerabschätzung ein BSC-Kanal (siehe Abschnitt 8.2) und statistisch unabhängige Bitfehler angenommen, so wie bei der Bewertung vergleichbarer Bussysteme unter Punkt 2.3.

Aufgrund der unterschiedlichen Codierung von „Fireworks-“ und Daten-„Frames“, besitzt „Basic-TTP/A“ zwei Restfehlerwahrscheinlichkeiten. Davon soll die „Fireworks“-Codierung zuerst behandelt werden. Nach Abschnitt 3.2.8 enthalten die „Fireworks-Frames“ einen (9,4)-CRC-Code mit einer Hamming-Distanz $d_{min} = 4$. Des Weiteren erkennt dieser Code alle ungeradzahligen Bitfehler. Es können also lediglich Vier-, Sechs- und Achtbitfehler ein Codewort unbemerkt verfälschen. Das gilt allerdings nicht für jede Bitfehlerkombination, weshalb eine Berechnung der Restfehlerwahrscheinlich mit der Binomialverteilung zu pessimistisch wäre. Bei dem verwendeten (9,4)-CRC-Code ist dafür hauptsächlich die „Burst“-Fehlererkennung verantwortlich. Sie gewährleistet, dass „Burst“-Fehler der Länge fünf sicher und darüber hinaus mit einer über 90 %-igen Wahrscheinlichkeit erkannt werden. Nun ist ein „Burst“-Fehler der Länge b mehr als b aufeinanderfolgende verfälschte Bits. Er kann auch unverfälschte Bits enthalten und wird trotzdem identifiziert. Die einzige Bedingung ist, dass ein „Burst“-Fehler mit einem verfälschten Bit beginnt und aufhört (Fehlermuster: 1xx..1). Wegen dieser Gründe, erkennt der (9,4)-CRC-Code auch zahlreiche Kombinationen von den 126 Mustern für Vierbitfehler, den 84 Mustern für Sechsbitfehler und den 9 Mustern für Achtbitfehler. Diese Reduktion des Binomialkoeffizients lässt sich nach [108] bei vielen Codes über den Faktor 2^{-k} (k: Anzahl der Kontrollstellen) approximieren, so dass für die Restfehlerwahrscheinlichkeit der „Fireworks-Frames“ ungefähr gilt:

$$P_{\text{Restfehler, Fireworks}} \approx 2^{-5} \cdot \sum_{e} \binom{9}{e} \cdot P_{\text{Bitfehler}}^{e} \cdot (1 - P_{\text{Bitfehler}})^{9-e} \qquad \text{für } e = 4, 6, 8$$

Formel 4.16: Restfehlerwahrscheinlichkeit der „Fireworks-Frames“ in „Basic-TTP/A“

Etwas einfacher gestaltet sich die Berechnung bzw. Abschätzung der Restfehlerwahrscheinlichkeit bei den Daten-„Frames“. Sie besitzen nach Abschnitt 3.2.7 nur eine Paritybitsicherung. Dementsprechend niedrig fällt ihre min. Codedistanz mit $d_{min} = 2$ aus. Ferner erkennt eine Paritybitsicherung alle ungeradzahligen, aber keine geradzahligen Bitverfälschungen. Dadurch fließen in die Restfehlerwahrscheinlichkeit der Daten-„Frames“ Zwei-, Vier-, Sechs- und Achtbitfehler ohne jegliche Reduktion ein. Obwohl das Ergebnis in Formel 4.17 dem in Formel 4.16 ähnelt, fällt die Restfehlerwahrscheinlichkeit der Daten-„Frames“ deutlich höher aus als die der „Fireworks-Frames“. Daran ist in erster Linie die geringere Codedistanz und in zweiter Linie die nicht vorhandene Fehlerreduktion schuld. Für einen Vergleich mit den Bussystemen aus Abschnitt 2.3, erfolgt eine Quantifizierung der beiden Formeln mit der inzwischen üblichen, durchschnittlichen Bitfehlerwahrscheinlichkeit $P_{Bitfehler} = 10^{-3}$. Dieser Wert ist zwar pessimistisch, beim Auftreten von „Burst“-Fehlern aber möglich (siehe Punkt 8.2). Es ergeben sich die folgenden Restfehlerwahrscheinlichkeiten: $P_{\text{Restfehler, Fireworks}} \approx 3{,}91 \cdot 10^{-12}$ und $P_{\text{Restfehler, Daten}} \approx 3{,}57 \cdot 10^{-5}$. Nach DIN 19244 (siehe Abschnitt 8.7) erfüllen die „Fireworks-Frames“ damit die Kriterien der Datenintegritätsklasse I2 (spontane Übertragung) und die typischerweise idempotenten Daten-„Frames“ die Kriterien der Datenintegritätsklasse I1 (zyklisch aufdatende Systeme). Auf diese Weise ist der besonderen Stellung der „Fireworks-Frames“ Rechnung getragen, so dass „Basic-TTP/A“ insgesamt in die Datenintegritätsklasse I1 fällt.

$$P_{\text{Restfehler, Daten}} = \sum_{e} \binom{9}{e} \cdot P_{\text{Bitfehler}}^{e} \cdot (1 - P_{\text{Bitfehler}})^{9-e} \qquad \text{für } e = 2, 4, 6, 8$$

Formel 4.17: Restfehlerwahrscheinlichkeit von Daten-„Frames“ in „Basic-TTP/A“

Allein wegen dieser Eigenschaft hebt sich „Basic-TTP/A“ von einigen anderen „low-cost“ Sensor/Aktorbussen ab (siehe Tabelle 4.1 auf S. 130). Weitere Pluspunkte sammelt „Basic-TTP/A“ aufgrund seiner Effizienz, impliziten Realzeitfähigkeit, durch das Mithören der Telegramme und seiner Erweiterbarkeit. In Abschnitt 6.3 wird z.B. gezeigt wie man die Fehlererkennungseigenschaften mit einfachen Mitteln verbessern kann, um die höchste Datenintegritätsklasse I3 (kritische Informationsübertragung) zu erreichen. Außerdem bietet die typische Idempotenz der Daten die Möglichkeit mit wenig Aufwand „Oversampling“ zu betreiben. Dadurch kann man z.B. eine Extrapolation für eine applikationsspezifische Plausibilitätskontrolle problemlos realisieren.

Dennoch eignet sich „Basic-TTP/A“ nicht für sicherheitskritische oder hochverfügbare Systeme. Es besitzt nämlich „Single-Points-of-Failures“, wie z.B. den „Master“ oder das Kabel. Folglich ist seine Verlässlichkeit als mittelmäßig einzustufen. Dadurch ist die Anwendungsdomäne auf das Gros der einfachen, kostengünstigen Systeme beschränkt. „Basic-TTP/A“ erfüllt aber wichtige Voraussetzungen für sicherheitskritische oder hochverfügbare Systeme, insbesondere wegen der impliziten Realzeitfähigkeit. Aus diesem Grund kann man es mittels einfacher Ergänzungen, die der TTP/A-Baustein „Extended-Reliability“ enthält, auch dafür qualifizieren. Mit anderen Sensor/Aktorbussen bzw. vergleichbaren Bussystemen ist das ebenso möglich. Allerdings mit sehr viel mehr Aufwand, da in der Regel Protokolländerungen notwendig sind. Zudem müssen Ergänzungen oft in der Applikation geschehen. Die dabei möglichen Probleme und Gefahren wurden in Kapitel 1 und 2 beschrieben.

4.3.4 Tabellarischer Vergleich

Bussystem	Zeitverhalten	min. Code-distanz	Restfehlerw. $P_{wg,Bit} = 10^{-3}$	Effizienz [%]	Baudrate [kBit/s]	Busphysik	max. Teil-nehmer	max. Länge	MAC[1] (2a)	LLC[2] (2b)	Kosten und Bemerkungen
ASI	impl. Zyklus; nicht konstant; nicht einstellbar; 1..10 ms	2	$< 10^{-12}$ *Klasse I3*[3]	10..15	≈ 167	2-Draht; spezielle APM-„Power-line“[4]	alte Spez. 31; neue Spez. 62	100 m ohne Repeater	M/S	Z	weniger günstig; ≈5..10 € pro IC; spez. ICs, Spule u. Kabel; Industrie; Sicherheitsvariante
ISO 9141	Teil der Applikation; lange Pausen mögl.; 100 ms...	2	$< 3 \cdot 10^{-5}$	10..15	0,01..10	1-Draht; NRZ; “Open Collector“	theo. 128 empf. 16	≈ 40 m abh. von C_{Bus}	M/S	Z	günstig; ein µC für Protokoll und Appl.; bis 0,5 € herab; alt; Busphysik aktuell; Kfz
LIN	nachr./ereignisgesteuert; lange Pausen mögl.; 100 ms...	2	$< 2 \cdot 10^{-4}$	10..20	1..20	1-Draht; NRZ; “Open Collector“	theo. 63 empf. 16	40 m	M/S	Z	günstig; ein LIN-µC für Protokoll und Appl.; bis 0,5 € herab; Kfz; neu; erste LIN-µCs
PPC	Teil der Applikation; 50 ms...	2	$< 9 \cdot 10^{-6}$	10..20	10..32	1-Draht; NRZ “Open Collector“	theo. 30 prakt. 20	10 m	M/S	Z	günstig bei großen Stk.; ASIC mit int. PLL-Oszi.; Kfz; i. d. Entwicklung
BST	In der Applikation; lange Pausen mögl.; 1..10 ms	≥ 4	$< 10^{-100}$ (FZ) $< 10^{-30}$ (ZV) $< 10^{-5}$ (NZ)[5]	15..25	31,25 125 250	2-Draht; spezielle SURFS-„Power-line“[6]	62, davon 12 mit Sprengkapseln	30 m	M/S	B	günstig bei hohen Stk.; ASIC; Kfz-„Airbags“ usw.; i. d. Erprobung; ISO-Norm geplant
CAN	„harte“ Realzeit nur mit TTCAN; 0,5..5 ms	2	$< 10^{-11}$ *Klasse I2*	5..10 (TTCAN: 10..15)	10..1000	2-Draht	> 32 (2032)	40... 1000 m	CSMA/CA	B	weniger günstig; ab 2..3 € pro IC; viele ICs; Kfz und Industrie; 2 Sicherheitsvarianten
„Basic-TTP/A“	impl. Zyklus; konst.; einstellbar; azykl. mögl.; 0,5..5 ms	≥ 2	$< 4 \cdot 10^{-5}$ *Klasse I1*	40..50	zur Zeit bis 625	mehrere z.Z. RS485, CAN, ISO 9141	theo. ≈ 16·254 prak. Grenze Busphysik	40... einige 100 m	TDMA + M/S	Z	günstig; ein µC für Protokoll und Appl.; bis 0,5 € pro herab; skalierbar; neu; Kfz und Indu.
„Basic-TTP/A“ + „Ext.-Reli.“	-||- 1..10 ms	≥ 6	$< 10^{-16}$ *Klasse I3*	15..25	-||-	-||-	-||-	-||-	-||-	-||-	ungefähr Verdopplung der „Slots“; Konzeptvorschlag

[1] MAC: „Media Access Control“; OSI-Schicht 2a
M/S: „Master/Slave“; TDMA: „Time Division Multiple Access“; CSMA/CA: „Carrier Sense Multiple Access/Collision Aviodance“

[2] LLC: „Logical Link Layer“; OSI-Schicht 2b
Z: zeichenorientiert; B: bitorientiert

[3] I1, I2, I3: Datenintegritätsklassen nach DIN 19244 (niedrig, mittel, hoch)

[4] APM-„Powerline“: Alternierende-Puls-Modulation speziell für ASI entwickelt; sin²-Impulse aus Manchester- Bits auf der Stromversorgung

[5] FZ: Fehlzündung; z.B. eines „Airbags“ während der Fahrt; ist am gefährlichsten, weil unfallprovozierend
ZV: Zündverfälschung; falsche oder andere „Airbags“ zünden; weniger gefährlich; Wiederholungen zünden geplanten „Airbag“
NZ: Normalzündung; alles passiert wie geplant; hohe Absicherung gegen Fehlzündungen senkt die Normalzündwahrscheinlichkeit

Tabelle 4.1: Vergleich von Sensor/Aktorbussen und ähnlichen Bussystemen

Die vorangehende Tabelle gestattet einen direkten Vergleich der Eigenschaften von „Basic-TTP/A“, „Basic-TTP/A“ mit dem Baustein „Extended-Reliability“ (Kapitel 6) und den bewerteten, vergleichbaren Bussystemen aus Abschnitt 2.3. Anhand insbesondere der Spalten 2 bis 6 und 12 werden die Vorteile des TTP/A-Busses deutlich: TTP/A ist realzeitfähig, schnell, effizient, kostengünstig, benötigt keine proprietären Bauteile und kann bezüglich der Redundanz für einfache und verlässliche Systeme ausgelegt werden.

4.4 Zusammenfassung

Eine große Herausforderung bei der Erarbeitung des Realisierungskonzepts für „Basic-TTP/A“ stellten die hohen Realzeitanforderungen, die Restriktionen nur Software und keine Spezialbauteile zu verwenden, die geringen Ressourcen von „low-cost“ µCs und die Diversität von vermeintlich gleichen Standardbauteilen dar. Wesentliche Probleme waren der UART-„Jitter“, die exakte Platzierung der „Fireworks-Frames“, die Synchronität der „Slot-Timer“, „Critical-Sections“, unterschiedlich lange Programmpfade, Latenz- und Befehlsausführungszeiten und die Baudraten-Abhängigkeit der Zeitfehler. Mit einem „Interleave“-Algorithmus, einer rechenzeitorientierten „Task“-Aufteilung, einer impliziten Synchronisation und einer Selbstjustierung ist es gelungen alle Schwierigkeiten zu überwinden. Vorausgesetzt werden eine unterbrechbare CPU, etwas flüchtiger und nicht flüchtiger Speicher, ein „Timer“ mit einer Minimalauflösung von $T_{Bit,\,M}/8$, ein Hardware-UART mit mindestens achtfacher Bitteilung oder eine entsprechende UART-Emulation und wenige I/O-Pins. Die meisten „low-cost“ µCs, DSPs o.Ä. und UARTs erfüllen diese Voraussetzungen, so dass man das beschriebene Realisierungskonzept als allgemeingültig ansehen darf.

Mit dem „Interleave“-Algorithmus und der rechenzeitorientierten „Task“-Aufteilung werden Abhängigkeiten zwischen den Ereignissen und unkritische Rechenzeiträume genutzt. Dadurch lassen sich die Reaktionszeiten stark reduzieren, unterschiedlich lange Programmpfade in zeitkritischen Situationen eliminieren und unnötige „Task“-Wechsel vermeiden. Für die exakte Platzierung der „Fireworks-Frames“ ist es weiterhin nötig, die Phase des „Slot-Timers“ im „Master“ automatisch zu justieren. Dieser Phasenabgleich bewirkt, dass weder „Critical-Sections“, noch die Baudraten-abhängigen, relativen Latenz- und Befehlsausführungszeitfehler zu einem „UART-Jitter“ bei den „Fireworks-Frames“ führen. Aufgrund ihrer exakten Platzierung, qualifizieren sich die „Fireworks-Frames“ als zeitliche Referenzmarken und schaffen die Voraussetzung für die implizite Synchronisation der „Slaves“. Diese umfasst eine Lagefehlerkorrektur des „Slot-Timers und, je nach Oszillatorgüte, eine Gangfehlerkorrektur des „Slot-Timers“ und UARTs. Zur Minimierung der Einflüsse von Latenz- und Befehlsausführungszeiten ist ein selbstjustierender Code in den „Slaves“ vorgesehen.

Es wurde theoretisch und praktisch gezeigt, dass die geforderten Zeitgenauigkeiten $|F_{UART}| \leq 1{,}84 \cdot 10^{-2}$ und $|F_{Timer}| \leq 1{,}52 \cdot 10^{-4}$ (maximale Rundenlänge) mit den geschilderten Lösungsstrategien, unter Berücksichtigung der Randbedingungen, eingehalten werden. Außerdem belegt eine Bewertung und ein tabellarischer Vergleich die Vorteile des neuen Sensor/Aktorbusses TTP/A, insbesondere hinsichtlich den Eigenschaften Realzeitverhalten, minimale Codedistanz, Restfehlerwahrscheinlichkeit, Übertragungseffizienz, maximale Baudrate und Kosten.

5 Umsetzung und Anwendung

5.1 Allgemeines

In diesem Kapitel wird die erste, universelle Implementierung von „Basic-TTP/A“ für einen „Master“ und einen „Slave“ vorgestellt. Anhand von zwei Beispielen werden des Weiteren die Anwendung und Funktionstüchtigkeit gezeigt. Die Implementierung ist ausschließlich in der Programmiersprache C gehalten, mit getrennten hardwareabhängigen und -unabhängigen Teilen. Auf diese Weise ist sie verhältnismäßig leicht portierbar, wobei die existierende C16x-Portierung als Vorlage dienen kann. Ferner lässt sich die Implementierung bezüglich der RODL-Größe, zusätzlicher „Tasks“, „Debug-Signale“ usw. über so genannte „Macros“ anpassen und optimieren. Die Quellcodedateien befinden sich, inklusive weiterer Informationen, auf der CD im Anhang.

5.2 Implementierung und Portierung

5.2.1 Verwendete Mikrocontroller (µCs)

Für die Erstellung und Erprobung der Erstimplementierung, fiel die Wahl des µCs zu Gunsten der C16x-Familie von Infineon aus. Diese µCs sind weit verbreitet – hauptsächlich Kfz-Steuergeräte – und haben bei der Entwicklung von „Basic-TTP/A“ einen großen Spielraum gelassen. Zudem gibt es kompatible Bausteine von anderen „Chip“-Herstellern (z.B. ST10*x*16*x* von SGS Thomson oder M16Cx von Mitsubishi), einen „IP-Core“ und in naher Zukunft ein verbessertes Derivat mit 20-fach höherer Leistung [132] [116]. Alle C16x-µCs besitzen einen Hardware-UART mit einer Baudrate bis 625 kBit/s bei einer typischen CPU-Taktrate von 20 MHz und mindestens drei „Timer“, von denen wenigstens einer „auto-reload“-fähig ist. Die Anzahl der I/O-Pins beläuft sich auf mehr als 50. Sie sind separat konfigurierbar und können entweder als digitale Ein-/Ausgänge fungieren oder andere Funktionen übernehmen. Darunter befinden Zählereingänge, „Timer“-Ausgänge, „Chip-Select“-Leitungen, „Interrupt“-Eingänge, eine synchrone serielle Schnittstellen (SPI), eine asynchrone serielle Schnittstellen (UART), ein „Watchdog“-Ausgang und selbstverständlich Adress-, Steuer- und Datenbusleitungen. Letztere erlauben den Anschluss von bis zu 16 MB externen Speicher mit einstellbarer Datenbusbreite (8 oder 16 Bit). Je nach Typ, können die C16x-µCs zusätzlich über einen A/D-Wandler, CAN-Bus, I²C-Bus, eine PWM, internes RAM (min. 1 KB), internes ROM, einen zweiten UART, eine „Realtime-Clock“ und ein „Power-Management“ verfügen.

Der Kern aller C16x-µCs, die nach dem klassischen „Von Neumann“-Prinzip arbeiten, ist eine 16 Bit-CISC/RISC-CPU. Laut [113] vereinen die C16x-µCs damit die Vorteile beider Verfahren: Einfache Operationen werden in einem Maschinenzyklus (100 ns @ 20 MHz) ausgeführt, währenddessen komplexe Operationen einen Befehl benötigen. Aus diesem Grund schwankt die Rechenleistung der C16x-

µCs. Sie beträgt mit einem 20 MHz-Takt max. 10 MIPS und basiert auf einer vierstufigen „Pipeline“ („Fetch“, „Decode“, „Execute“, „Writeback“) und einem „Jump-Cache“. Je einfacher ein Programm ist, desto näher kommt die tatsächliche Rechenleistung dem Maximalwert. Zu den Besonderheiten gehören eine „Multiply/Division“-Einheit und das „Interrupt“-System. Die „Multiply/Division“-Einheit führt eine „16 Bit·16 Bit-Multiplikation“ in 5 Maschinenzyklen und eine „32 Bit/16 Bit“-Division in 10 Maschinenzyklen aus. Das „Interrupt“-System erlaubt zum einen die Priorisierung von „Interrupts“. Zum anderen gestattet es schnelle Kontextwechsel, mittels Umschalten von ganzen Registerbänken. Aus diesem Grund hat ein C16x-µC keine festen „General-Purpose-Register“, sondern er adressiert diese über einen so genannten „Context-Pointer“ im internen RAM. Das Umprogrammieren des „Context-Pointers“ erfolgt mit einem einfachen Befehl und dauert folglich nur 100 ns (ein Maschinenzyklus). Allerdings darf man diese Zeit nicht mit der „Interrupt“-Latenzzeit verwechseln, sie ist ein Teil davon. Die „Interrupt“-Latenzzeit kann zwischen 250 ns und ca. 1 µs liegen, je nach Art des „Interrupts“ („internal“, „external“, „fast“), Belegung der „Pipeline“ und Anzahl der „Wait-States“.

Bezogen auf die Rechenleistungen heutiger µCs, bewegen sich die C16x-µCs im unteren Leistungssegment[111]. Das soll ein Vergleich mit ein paar typischen „low-cost“ µCs zeigen. Von Atmel gibt es die 8 Bit „low-cost“ Bausteine der AVR-Reihe. Ein Repräsentant ist z.B. der AT90S2313 [119]. Er erreicht mit $f_{CPU} = 20$ MHz ebenfalls 10 MIPS. Die PIC-Familie von Microchip ist etwas leistungsschwächer. Mit einem 20 MHz-Oszillator schafft z.B. der PIC16F627 ungefähr 5 MIPS [118]. Ein großes Spektrum hat sich auch um den bekannten „low-cost“ µC 8051 gebildet. Philips bietet z.B. das Derivat 87LPC767 an, das mit $f_{CPU} = 20$ MHz bis zu 3,3 MIPS stark ist [124]. Für einen fairen Vergleich muss man die Rechenleistung dieses Bausteins etwas höher bewerten, da der 8051 eine CISC-CPU besitzt und die anderen beiden „low-cost“ µCs ein RISC- bzw. RISC-ähnliche CPU. Es gibt aber auch wesentlich schnellere 8051-Derivate. Die Firma Dallas bietet z.B. 100 MHz-Varianten an, die hochgerechnet über ca. 33 MIPS Rechenleistung verfügen. So gesehen liefert die C16x-Portierung von „Basic-TTP/A“, neben den Daten über den Ressourcenbedarf, gute Anhaltswerte für die Rechenzeiten, die CPU-Belastung und die max. Baudrate. Preislich sind die C16x-µCs für „Slaves“ jedoch weniger attraktiv. Ihre reichliche Ausstattung schlägt sich in einem Preis von etwa 5..10 € nieder. Die eben genannten „low-cost“ µCs eignen sich dafür besser. Sie kosten ungefähr zwischen 0,5..2 € pro Baustein[112]. Für einen „Master“ ist ein C16x-µC, trotz der moderaten Rechenleistung, aber durchaus interessant. Besonders seine vielen Schnittstellen (CAN, SPI, I^2C, zweiter UART) befähigen ihn zu einem „Master“ mit „Gateway“-Funktionalität, so wie im Demonetz gezeigt. Und dank des Konzepts aus Kapitel 4, reicht das Baudraten-Spektrum mit bis zu 625 kBit/s auch vollends aus.

Als Vertreter der C16x-Reihe kamen der C165 [113] und der 80C166 [114] zum Einsatz. Sie befinden sich hinsichtlich Funktionen, Ressourcen und Kosten im Mittelfeld dieser Familie. Die Entscheidung zwei unterschiedliche µCs zu verwenden hatte folgenden Grund: Der 80C166 ist der einzige C16x-µC mit zwei UARTs. Er war bei der Entwicklung von „Basic-TTP/A“ sehr hilfreich, weil alle preiswerten „Debugger“ für diese µCs, ebenso wie „Basic-TTP/A“, einen UART benötigen. Allerdings ist der 80C166 der älteste µC aus der C16x-Familie und enthält folglich einige wichtige Neuerungen nicht. Deshalb verdrängen ihn seine Nachfolger, wie z.B. der C165, zunehmend, so dass mit der Einstellung seiner Produktion zu rechnen ist. Anstelle des 80C166 hätte man den C165 natürlich auch um einen externen UART erweitern können. Der Hardware-Aufwand wäre ähnlich gewesen, der Software-Aufwand hingegen groß. „Basic-TTP/A“ sollte auf jeden Fall mit dem internen UART arbeiten, um das

[111] Infineon möchte mit dem bis zu 20-mal stärkeren Nachfolger C166S V2 wieder einen µC im mittleren Leistungssegment anbieten – einst gehörten die seit vielen Jahren existierenden C16x-µCs zu diesem Segment.

[112] Sämtliche Preisangaben sind grobe Anhaltswerte, da je nach Typ, Temperaturbereich und insbesondere Stückzahlen große Schwankungen nach oben und unten möglich sind.

Projektziel zu treffen. Aus diesem Grund hätte der „Debugger“ über den externen UART laufen müssen. Der so genannten „Debug-Monitor“ ist jedoch für den internen UART ausgelegt, so dass aufwändige Modifikationen an ihm notwendig gewesen wären. Das war ein Grund für den 80C166. Ein Zweiter war die einfache Erprobung der Portierbarkeit. Wie erwähnt, unterscheidet sich der 80C166 am meisten von den Mitgliedern der C16x-Familie. Seine UARTs unterstützen z.B. nur „Even-Parity“, „Basic-TTP/A“ benötigt aber auch „Odd-Parity“. Da dieses Problem bei ein paar „low-cost“ µCs, wie z.B. dem ST62x18 [118], ebenso auftritt, enthält die 80C166-Portierung hierfür einen Lösungsvorschlag. Und nicht zuletzt, konnten auf diese Weise die „Project-“ und „Make-Files“ getestet und korrigiert werden.

5.2.2 „Target-Boards“

Auf Basis der ausgewählten C16x-µCs wurden und werden verschiedene, universelle „Basic-TTP/A“-Knoten entworfen und gebaut. Diese so genannten „Target-Boards“ sind in Bild 5.1 abgelichtet. Das „Target-Board“ mit dem µC 80C166 trägt den Namen C166node und die „Target-Boards“ mit den µCs C165 die Namen C165node x. Mit dem Platzhalter x werden die Versionen bzw. Ausprägungen der unterschiedlichen C165nodes gekennzeichnet. Links in Bild 5.1 ist das erste, streichholzschachtelgroße „Target-Board“, der C165node I zu sehen. Er wurde von der TU München entwickelt und vom Projektpartner SiemensVDO gefertigt. Aufgrund des geplanten Einsatzes in einem Demonetz (siehe Abschnitt 5.4), wurde der C165node I in seiner Größe optimiert. Der „Sandwich“ besteht aus zwei beidseitig vollbestückten Platinen, der oberen Basisplatine und der unteren Controllerplatine. Wegen seiner Funktionstüchtigkeit und Robustheit, avancierte dieser Knoten unerwartet auch zur Hardware-Basis für die Projektpartner. SiemensVDO wollte mit dem C165node I den „High-Speed-Physical-Layer“ testen und die Universität Stuttgart ihre Internet-Anbindung.

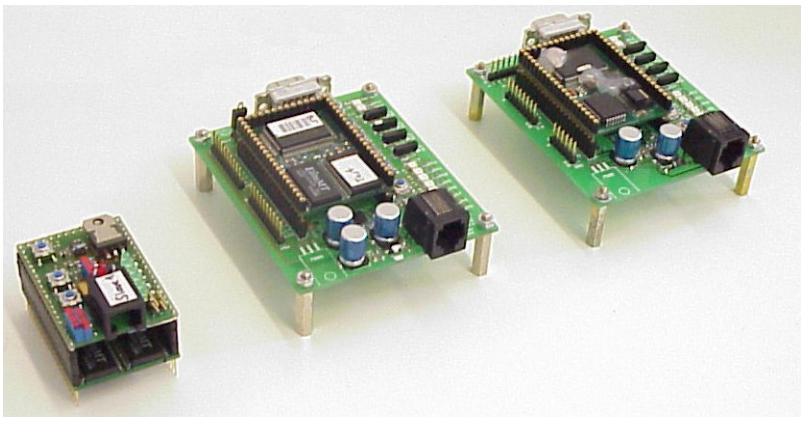


Bild 5.1: C165node I (links), C165node II (Mitte) und C166node (rechts)

Es zeigte es sich jedoch, dass die Basisplatine des C165nodes I zu viele Pins des µCs für die teils umfangreichen Erweiterungen belegte. Aus diesem Grund hat die TU München aus dem C165node I den C165node II abgeleitet. Er ist in der Mitte von Bild 5.1 dargestellt. Die Basisplatine (unten) des C165nodes II ist deutlich größer als die seines Vorgängers. Auf ihr befindet sich die kleine Controllerplatine als „Piggyback“-Modul. Zwar ist der C165node II nicht so smart wie der C165node I, dafür bietet seine Basisplatine, auf der nahezu freien Unterseite, reichlich Platz für weitere „Chips“ und hält außerdem die nötigen µC-Signale parat. Ferner wurde der C165node II etwas erweitert, um die Software-Entwicklungszyklen zu beschleunigen und seine Weiterentwicklung zu unterstützen. Aus dem

C165node II entstand mittlerweile der C165node III von SiemensVDO mit dem in Abschnitt 3.3.3 kurz beschriebenen synchronen „High-Speed-Physical-Layer“. Zudem arbeitet die Universität Stuttgart gerade am C165node IV, der mit einem „Ethernet-Chip“ ausgestattet und einem „Web-Server“ auf dem C165-µC die direkte Internet-Anbindung ermöglichen soll.

Alle „Target-Boards“ besitzen eine RS485-Schnittstelle für „Basic-TTP/A“. Der Busanschluss erfolgt über sechspolige RJ11-Buchsen[113], die in Bild 5.1 als schwarze Quader auf den Vorderseiten der „Target-Boards“ zu sehen sind. Die Pinbelegung ist unter Punkt 3.3.2 dokumentiert. Außerdem verfügen der C165node II und der C166node über je eine RS232-Schnittstelle für die Kommunikation mit z.B. einem PC. Der Zugang befindet sich auf den Rückseiten der beiden „Target-Boards“ in Form von neunpoligen Sub-D-Buchsen[114]. Beim C166node sind die RS485- und RS232-Schnittstelle, wegen der beiden UARTs in dem µC, unabhängig, währenddessen sie beim C165node II zwar elektrisch entkoppelt sind, aber einen gemeinsamen UART benutzen. Deshalb eignet sich der C166node vor allem zum „Debuggen“, als Busmonitor und als „Master“ mit „RS232-Gateway“. Der C165node II kann zumindest komfortabel programmiert werden und in seinen Ausbaustufen C165node III und IV von einer ähnlichen Funktionalität profitieren.

Ein paar Details zu den C165nodes

Das Blockschaltbild in Bild 5.2 zeigt den prinzipiellen Aufbau der beiden C165nodes. Die grau unterlegten Teile kennzeichnen die Unterschiede des C165nodes II gegenüber dem C165node I. Beide Knoten besitzen einige digitale Ein-/Ausgänge, die für Beobachtungs- und Testzwecke mit LEDs, Schaltern und Tastern verbunden sind. Zudem haben sie einen 12 Bit-ADC mit 500 kSamples/s, „Sample & Hold“-Stufe und ±5 V-Eingang, einen 200 kHz-schnellen 12 Bit-DAC mit ±5 V-Ausgang, entsprechende Filter, Schutzbeschaltungen und eine RS485-Schnittstelle für „Basic-TTP/A“. Aus Platzgründen verfügt der C165node I über einen linearen Spannungsregler, wohingegen der C165node II mit einem verlustarmen Schaltwandler arbeitet.

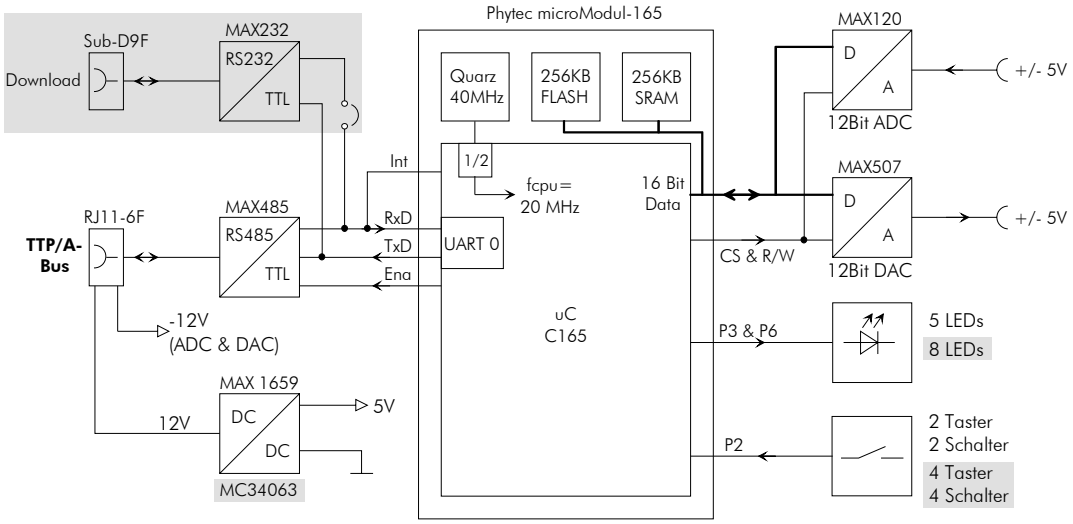


Bild 5.2: Blockschaltbild für den C165node I und II

[113] Diese Art der Steckverbindung wurde durch die Telephonie (vierpolig) und das 10BaseT-Ethernet (achtpolig) weit verbreitet und sehr günstig.

[114] Die Kreuzung der Signale TxD und RxD wird auf den „Target-Boards“ durchgeführt (1:1-Kabel).

Darüber hinaus besteht für „Monitoring“-Zwecke und Versuche mit Software-UARTs eine Verbindung zwischen der RxD-Leitung und einem „Interrupt“-Pin. Zuzüglich der RS232-Option des C165nodes II, sind die beiden Basisplatinen damit im Großen und Ganzen beschrieben. Auf den Controllerplatinen der Firma Phytec befinden sich in erster Linie der C165-µC, ein Quarzoszillator, 256 KB SRAM und 256 KB FLASH. Die Speicheranbindung ist 16 Bit breit und benötigt keine „Wait-States“. Weitere Informationen sind dem Schaltplan aus Abschnitt 8.10 zu entnehmen.

Ein paar Details zum C166node

Auf der Basisplatine des C166nodes sind ein Schaltwandler, LEDs, Schalter, Taster, ein OPV mit Tiefpass für einen langsamen PWM-DAC, Filter, Schutzbeschaltungen, eine RS485-Schnittstelle für „Basic-TTP/A“ und eine RS232-Schnittstelle untergebracht. Das ebenfalls von der Firma Phytec bezogene Controllermodul enthält den 80C166-µC, einen Quarzoszillator, 128 KB SRAM, 128 KB FLASH und einen RS485/TTL-Konverter. Die Speicheranbindung kommt ohne „Wait-States“ aus und ist zu Gunsten der I/O-Pins nur 8 Bit breit. Auf die Ausführungszeiten von „Basic-TTP/A“ haben die Bytezugriffe kaum Einfluss, weil die Implementierung hauptsächlich mit Bytes arbeitet. Die separate Parityberechnung macht sich hingegen deutlich bemerkbar. Je nach Knotentyp („Master“, Sensor-„Slave“, Aktor-„Slave“), reduziert sie die max. Baudrate von „Basic-TTP/A“ um den Faktor zwei bis drei. Außerdem wurde auch beim C166node die RxD-Leitung des „Basic-TTP/A“-UARTs mit einem „Interrupt“-Eingang verschaltet, um Tests mit Software-UARTs durchführen zu können oder für „Monitoring“-Zwecke. Wie das Blockschaltbild in Bild 5.3 vermuten lässt, muss diese Maßnahme allerdings händisch geschehen. Ein Schaltplan zum C166node enthält der Anhang (siehe Abschnitt 8.11).

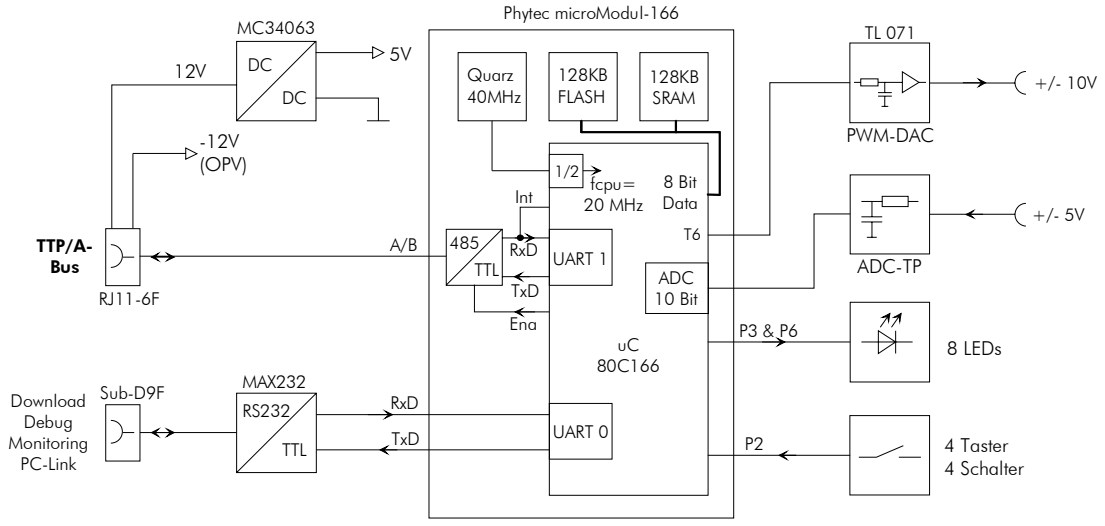


Bild 5.3: Blockschaltbild des C166nodes

5.2.3 Entwicklungsumgebung

Werkzeuge

Die Programmierung der „Target-Boards“ erfolgte mit den Software-Werkzeugen der Firma Tasking und der Firma Phytec auf einem Standard-PC (Celeron 400) mit Windows NT 4.0. Zu den Tasking-Werkzeugen gehört ein anpasster Codewright-Editor mit „Makefile-Generator“, die so genannte „Tasking Embedded Environment (EDE) v2.2 r3“. Außerdem umfassen sie die eigentliche „Tool-

Chain“ mit der Bezeichnung „Tasking C for C166/ST10 v6.0 r4”, bestehend aus einem „C-Cross-Compiler“, „Cross-Assembler“ und „Linker/Locator“. Ein „Debugger“ mit Simulator namens „Cross-View Pro v6.0 r4“ rundet die Tasking-Werkzeuge ab. Da es sich um einen „Software-Debugger“ handelt, muss man für eine Fehlersuche zusätzlich einen so genannten „Debug-Monitor“ auf das entsprechende „Target-Board“ laden und ausführen. Sein Ressourcenbedarf und die Interaktionen mit dem eigentlichen Programm erfordern ein separates Übersetzen und Binden des Quellcodes. In diesem als „Debug-Mode“ bezeichneten Zustand läuft das Programm unter der Kontrolle des „Debuggers“ nichtresident im SRAM. Hierfür kommuniziert der „Debugger“ auf dem PC mit dem „Debug-Monitor“ auf dem „Target-Board“ über eine serielle Standardschnittstelle (RS232, max. 38,4 kBaud). Sobald das Programm fehlerfrei arbeitet, muss es im „Release-Mode“ neu übersetzt und gebunden werden. In diesem Modus enthält es keinen „Debug-Code“ und ist, in Form einer Datei im „Intel-Hex-Format“, für die residente Speicherung im FLASH des „Target-Boards“ vorbereitet. Die FLASH-Programmierung (Brennen) wurde, ebenfalls über die serielle Standardschnittstelle, mit dem Software-Werkzeug „Flash Terminal V1.43“ der Firma Phytec durchgeführt. Um sowohl das eine als auch das andere nachvollziehen zu können, befinden sich auf der CD unter Abschnitt 8.13 die entsprechenden „Debug“- und „Release“-Dateien.

Speichermodell

Ferner wurde bei der Programmierung ausschließlich das „Small-Memory-Model“ verwendet. Dieses Speichermodell wird häufig angewandt, weil es, z.B. im Vergleich zum „Large-Memory-Model“, den direkt adressierbaren Speicher ohne zusätzliche Adressrechnung anspricht, aber auch den Zugriff auf Daten im gesamten Speicherbereich zulässt. Bei den C16x-µCs ist der direkt adressierbare Speicher, gemäß ihrer 16 Bit-Architektur, 64 KB groß, was für „Basic-TTP/A“ und eine Applikation in der Regel weit mehr als genug ist. Mit dem „Small-Memory-Model“ muss man jedoch auf eine Besonderheit bzw. Eigenheit der C16x-µCs Rücksicht nehmen. Sie soll am Beispiel des C165nodes II erklärt werden. Dazu ist in Tabelle 5.1 die Speicherbelegung dieses „Target-Boards“ in groben Zügen wiedergegeben.

Adresse	Speichertyp	Größe
0x00 0000 - 0x00 EFFF	FLASH	60 KB
0x00 F000 - 0x00 FFFF	interner RAM und Register	4 KB
0x01 0000 - 0x03 FFFF	FLASH	192 KB
0x04 0000 - 0x07 FFFF	SRAM	256 KB
0x10 0000 - 0x10 0FFF[115]	D/A- und A/D-Wandler	4 KB

Tabelle 5.1: „Memory-Map“ des C165nodes II

Für die Erweiterung ihres Adressraums besitzen die C16x-µCs ein 8 Bit CSP-Register („Code-Segment-Pointer“) und vier 10 Bit DPP-Register (Data-Page-Pointer). Damit können sie insgesamt 16 MB Speicher verwalten, wobei die Adressierung von Code und Daten unterschiedlich ist. Mit dem CSP-Register wird der Adressraum für den Code in 256 Segmente à 64 KB aufgeteilt. Folglich setzt sich eine 24 Bit breite Codeadresse aus dem Inhalt des 8 Bit CSP-Registers gefolgt vom Inhalt des 16 Bit IP-Registers (Instruction-Pointer) zusammen. Im „Small-Memory-Model“ ist der Code auf 64 KB

[115] Der A/D- und D/A-Wandler befinden sich oberhalb von der 1 MB-Adresse, weil es die gleichen Controllermodule auch mit 512 KB FLASH und 512 KB SRAM gibt.

beschränkt und befindet sich zumeist im Segment 0 (Adr. 0x00 0000 - 0x00 FFFF), so wie im Fall C165node II. Das Setzen des CSP-Registers übernimmt der „Compiler“ anhand der „Memory-Map“.

Bei den Daten ist die Sachlage nicht so einfach. Mit dem „Small-Memory-Model“ können auch hier 64 KB direkt adressiert werden. Allerdings müssen diese 64 KB nicht am Stück vorliegen, so wie beim Code. Denn die oberen zwei Bits der 16 Bit-Datenadresse wählen eines der vier DPP-Register (DPP 0..DPP 3) aus. Zur Bildung der 24 Bit-Absolutadresse werden an den 10 Bit-Eintrag des entsprechenden DPP-Registers die 14 Restbits der Datenadresse gehängt. Auf diese Weise entstehen $2^{10} = 1024$ so genannter „Data-Pages“ à 2^{14} KB = 16 KB. Mittels Programmierung der DPP-Register können im „Small-Memory-Model“ vier beliebige „Data-Pages“ zu einem quasi direkt adressierbaren, 64 KB großen Datenbereich zusammengefasst werden. Trotz Eingabe einer „Memory-Map“, kümmern sich die „Compiler“ um diese Aufgabe leider nicht. Es liegt also in der Verantwortung des Programmierers die DPP-Register korrekt zu Setzen.

Nach Tabelle 5.1, kommen im Beispiel C165node II hauptsächlich die „Data-Pages“ im Adressfenster 0x04 0000 - 0x07 FFFF (SRAM) in Frage. Sie besitzen die Nummern 16..31 und dürfen den DPP-Registern DPP 0..DPP 2 willkürlich zugeordnet werden. Das DPP-Register DPP 3 muss jedoch auf die „Data-Page“ drei zeigen (Adr. 0x00 C000 - 0x00 FFFF), damit der internen RAM und die Register adressierbar sind. Auf diese Weise entsteht eine Überlappung mit dem FLASH-Speicher, die mit der „Memory-Map“ auf das tatsächlich benötigte Adressfenster 0x00 F000 - 0x00 FFFF reduziert werden sollte. Diese Besonderheit bzw. Eigenheit schränkt, beim C165nodes II mit „Small-Memory-Model“, den direkt adressierbaren Codespeicher auf 60 KB und den direkt adressierbaren Datenspeicher auf 48 KB ein, zuzüglich 2 KB internem RAM. Daten, die außerhalb dieses Bereichs liegen, wie z.B. der A/D- und D/A-Wandler (siehe Tabelle 5.1), müssen über „Far“-Zugriffe adressiert werden. Wegen der expliziten Berechnung der 24 Bit-Adresse, dauern „Far“-Zugriffe im „Small-Memory-Model“ allerdings etwas länger.

5.2.4 Code-Aufteilung

Die Implementierung von „Basic-TTP/A“ ist, wie TTP/A selbst, nach Manier der Betriebssysteme in Bausteine gegliedert. Sie umfasst den hardwareunabhängigen Baustein „Basic-TTP/A-Kernel“ und die beiden hardwareabhängigen Bausteine „CPU-Port“ und „Board-Support-Package“ (siehe Bild 5.4). Auf diese Weise wird die Portierbarkeit von „Basic-TTP/A“ optimal unterstützt, weil die Suche nach hardwarespezifischem Code entfällt. Wenn ein anderer µC verwendet werden soll, benötigt man nur einen entsprechenden „CPU-Port“ und keine neue Implementierung. Das Gleiche gilt natürlich auch für ein „Target-Board“. Außerdem bleibt mit dieser Technik der eigentliche Kern, der Baustein „Basic-TTP/A-Kernel“, in allen Portierungen konsistent. Für die notwendige Entkopplung der Bausteine sorgen „Macros“.

„Basic-TTP/A-Kernel“

Wie Bild 5.4 zeigt, umfasst der hardwareunabhängige Baustein „Basic-TTP/A-Kernel“ sieben Dateien mit dem Namenspräfix „TTPa_“. Die Dateien „TTPa_Master.c“ und „TTPa_Master.h“ bilden den Hauptteil des „Masters“ und analog die Dateien „TTPa_Slave.c“ und „TTPa_Slave.h“ den Hauptteil des „Slaves“. Ferner enthält die Datei „TTPa_Common.c“ identische Codestücke für den „Master“ und den „Slave“ und die Datei „TTPa_Etc.h“, für die bessere Lesbarkeit des Quellcodes, TTP/A-spezifische Definitionen. An diesen sechs Dateien wird in der Regel nichts verändert, wohingegen die siebte Datei „TTPa_Config.h“ für die Parametrierung des „Masters“ und „Slaves“ editiert werden muss. Zu den wichtigsten Parametern gehören die Anzahl der benötigten Runden und „Slots“. Sie bestimmen den Speicherbedarf der RODL des jeweiligen Knotens. Des Weiteren kann der Applika-

tionsmodus eingestellt werden. Mit ihm legt man fest, ob eine Synchronisations-„Task“ eingerichtet wird oder nicht. Details hierzu sind im Quellcode auf der CD im Anhang ebenso nachzulesen wie alle weiteren Parameter. Aufgrund der Einstellmöglichkeiten, lassen sich der „Master“ und vor allem der „Slave“ hinsichtlich des Speicherbedarfs optimieren. Deshalb eignet sich diese Implementierung auch für ressourcenschwächere Knoten als die C16xnodes. Ein simpler „Slave“ z.B. benötigt 1,5..2 KB Speicher, wohingegen ein aufwändiger „Slave“ durchaus 20 KB Speicher und mehr belegen kann („Basic-TTP/A“ + Applikation). Es sei angemerkt, dass die „Master“- und „Slave“-Implementierung universell und nicht codeoptimiert sind. Schätzungsweise lässt sich ihr Speicherbedarf durch Spezialisierung und Optimierung um bis zu 20 % reduzieren, was in erster Linie für den „Slave“ interessant ist.

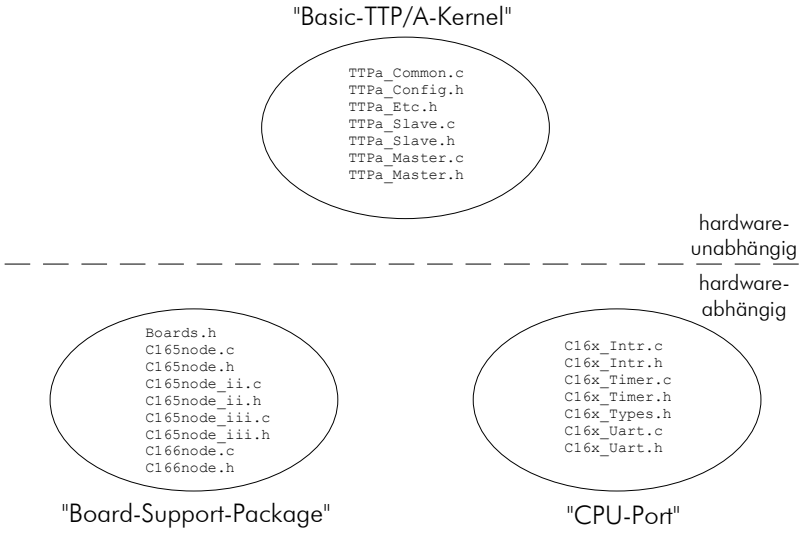


Bild 5.4: Code-Aufteilung der „Basic-TTP/A“-Implementierung

„CPU-Port“

Der „CPU-Port“ umfasst sämtliche µC-spezifischen Teile von „Basic-TTP/A“. Bisher existiert dieser Baustein für die µCs der C16x-Familie. Weitere so genannte Portierungen sollen im Forschungsvorhaben „Smart-Transducers“ (siehe Kapitel 7) entstehen. Zur Differenzierung werden die Dateien einer Portierung nach ihrem µC benannt. Folglich beginnen sämtliche Dateinamen der C16x-Portierung mit der Bezeichnung „C16x_“. Dem Bild 5.4 kann man entnehmen, dass die C16x-Portierung aus sieben Dateien besteht. Sie bilden drei so genannte Software-Module (kurz Module), die für einen „Master“ und „Slave“ gleich sind. Das erste Modul betrifft den UART. Zu ihm gehören die Dateien „C16x_UART.c“ und „C16x_UART.h“. Sie enthalten den Code für die Initialisierung und den Betrieb des UARTs in einem C16x-µC. Analog dazu setzt sich das „Timer“-Modul aus den Dateien „C16x_Timer.c“ und „C16x_Timer.h“ zusammen und das „Interrupt“-Modul aus den Dateien „C16x_Intr.c“ und „C16x_Intr.h“. Und schließlich die Datei „C16x_Types.h“. In ihr findet man eigene Definitionen für die üblichen Datentypen. Das mag auf den ersten Blick überflüssig erscheinen, ist für eine gute Portierbarkeit aber notwendig. Die „Compiler“-Hersteller handhaben die üblichen Datentypen nämlich unterschiedlich. Ein „Compiler“ codiert eine „Integer-Variable“ als 16 Bit-Wert und ein anderer „Compiler“ als 32 Bit-Wert. Ähnliches findet sich auch bei den Datentypen „Word“, „Long“, „Char“ usw. Wenn man diesen Punkt außer Acht lässt, kann nicht nur Speicher vergeudet werden, sondern es können auch Berechnungen falsche Ergebnisse liefern. Außerdem würde die Por-

tierbarkeit darunter leiden, da die „Basic-TTP/A“-Bausteine nicht entkoppelt wären und bei jeder Portierung überarbeitet werden müssten. So gesehen ist der Aufwand eigene Datentypen zu definieren deutlich geringer und weniger fehleranfällig.

„Board-Support-Package“

Der letzte Baustein für „Basic-TTP/A“ ist das „Board-Support-Package“ (kurz BSP). Ein BSP enthält sinngemäß alle Codestücke und Definitionen für die Initialisierung eines bestimmten „Target-Boards“ und dessen Betrieb. Da im Laufe dieser Arbeit mehrere „Target-Boards“ entstanden, existieren bereits entsprechende viele BSPs. Von der TU München stammen die drei BSPs für die „Target-Boards“ C166node, C165node I und C165node II (siehe Punkt 5.2.2) und von SiemensVDO das BSP für den C165node III (High-Speed-Physical-Layer). Mit dem BSP für den C165node IV (Ethernet) von der Uni Stuttgart ist dieses Jahr zu rechnen. Ein BSP besteht aus zwei Dateien, deren Namen die Bezeichnung des jeweiligen „Target-Boards“ sind. Im Fall C165node II heißen diese Dateien also „C165node_II.c“ und „C165node_II.h“. Neben den BSP-Dateien existiert die Datei „Boards.h“. Sie ordnet „Basic-TTP/A“ die erforderlichen Ressourcen zu, was einen schnellen Überblick gestattet.

5.2.5 „C16x-Master“

Baudrate und Rechenlast

Der „Basic-TTP/A-Master“ für die C16x-µCs (kurz „C16x-Master“) ist nach den Ausführungen von Kapitel 4 implementiert. Er berücksichtigt die günstige Rechenzeitverteilung, arbeitet mit dem „Interleave“-Algorithmus und führt einen Phasenabgleich zwischen dem „Slot-Timer“ und UART durch. Die Rundensteuerung ist mit einem konfigurierbaren Software-FIFO ausgestattet, um einer Applikation bei hohen Baudraten einen Rechenzeitgewinn zu verschaffen. Weiterhin sind sämtliche „Basic-TTP/A-Tasks“, inklusive der Synchronisations- und „Error-Task“ als ISRs realisiert. Aufgrund dieser Technik liegt die max. Baudrate eines „C16x-Masters“ mit 20 MHz CPU-Takt bei 625 kBit/s. Ein „Fireworks-Jitter“ entsteht nicht, weil der Phasenabgleich einen min. Spielraum von $½ \cdot T_{Bit,\,min} = 800$ ns garantiert und der „C16x-Master“ max. 500 ns in Anspruch nimmt (100 ns pro Befehl). Für die zufälligen Sendeverzögerungen sind implizite „Critical-Sections“ und die vierstufige „C16x-Pipeline“ verantwortlich. Die impliziten „Critical-Sections“ sind eine Folge des so genannten „Extend-Addressing-Schemes“ [113], das ein C16x-µC für Zugriffe auf „Far“-Daten und „Extended-Special-Function-Registers“ anwendet. Theoretisch können sie fünf Maschinenbefehle lang sein[116]. Eine Analyse der Maschinencodes von einfachen und komplexen Applikationen (Kfz-Blinker, Regelung, „IFS/XML-Gateway“...) hat jedoch ergeben, dass sie sich in der Praxis nur über zwei Maschinenbefehle erstrecken. Stellvertretend für den Zugriff auf ein „Extended-Special-Function-Register“ und ein „Far“-Datum zeigt Bild 5.5 je eine Maschinencodesequenz aus den Anwendungsbeispielen unter Punkt 5.3 und Punkt 5.4.

```
...
EXTR #01h              ;two atomic instructions to modify
AND  ODP3,#0FBFFh      ;extended special function register ODP3
...
EXTP R13,#01h          ;two atomic instructions to read far data
MOV  R14,[R12]         ;from [R13:R12] and save it in R14
...
```

Bild 5.5: Beispiele für implizite „Critical-Sections“ im C16x-Maschinencode

[116] Ein einleitender Befehl (ATOMIC, EXTR, EXTP, EXTS, EXTPR, EXTSR) plus max. vier weitere Befehle.

Demnach haben die impliziten „Critical-Sections“ des „C16x-Masters“ am genannten Sendeverzug einen Anteil von max. 200 ns. Die restlichen 300 ns sind der vierstufigen „C16x-Pipeline“ zuzurechnen. Je nachdem wie sie gerade belegt ist, dauert die Latenzzeit eines internen „Interrupts“, zu denen der „Timer-Interrupt“ gehört, 300 bis 600 ns [113].

Wenn man die Baudratenbeschränkung des C16x-UARTs auf 625 kBit/s (f_{CPU} = 20 MHz) einmal außer Acht lässt, läge die max. Baudrate des „C16x-Masters“ bei $2 \cdot (500\ ns)^{-1}$ = 1 MBit/s. Darüber könnten die „Fireworks-Frames“ zu „jittern“ beginnen, weil der Spielraum durch den Phasenabgleich mit 1 MBit/s aufgebraucht wäre – als Zeitreferenz spielen für einen „Master“ anderweitige Zeitfehler (z.B. „Frame“-Dauer) keine Rolle. Eine ähnlich hohe theoretische Grenze ergibt sich auch wegen der Rechenzeitrestriktionen für die „Basic-TTP/A-Tasks“ des „C16x-Masters“ (siehe Bild 4.13 in Abschnitt 4.2.2). Doch selbst wenn der C16x-UART mehr zu leisten im Stande wäre, eine Baudrate über 625 kBit/s hätte mit einem C16x-µC wenig Sinn. In Abschnitt 4.2.2 wurden nämlich ebenso Messungen und Berechungen zur relativen Rechenlast des „C16x-Masters“ angestellt. Sie belegen eine CPU-Last von 53,6 % bei 625 kBit/s und eine ideelle CPU-Last von 85,8 % bei 1 MBit/s. Aus den praktischen Beispielen ist bekannt, dass einfache Applikationen (Blinker, Taster, Fühler...) bei 625 kBit/s mit der verbleibenden, durchschnittlichen Rechenleistung von knapp 5 MIPS gut zurecht kommen[117]. Hingegen treten in aufwändigen Applikationen („IFS/XML-Gateway“, verteilte Regelung, verteilte Textausgabe über printf()...) erste Engpässe auf. Deshalb stellt die max. Baudrate des C16x-UARTs auch eine sinnvolle Obergrenze für den „C16x-Master“ dar. Aus diesen Kennwerten lässt sich eine rechenleistungsbezogene max. Baudrate von 50..60 kBaud/MIPS ableiten. Da ein C16x-µC ein typischer µC ist, kann sie für erste Abschätzungen vernünftiger Baudratenobergrenzen in zukünftigen Portierungen dienen.

Speicherbedarf

Laut „Compiler-Report“ beläuft sich der Speicherbedarf ohne Applikation und ohne Optimierung auf knapp 1,5 KB für den Code und bei einer max. großen RODL auf ca. 8 KB für die Daten. Zusätzlich werden ungefähr 20 Bytes für Zähler, Puffer und andere Variablen benötigt. Die RODL ist als zweidimensionales „Array“ realisiert worden, mit der Rundennummer und der „Slot“-Nummer als Indizes. Jeder Eintrag des „Arrays“ enthält ein Byte für das Datum und ein Byte für das „Slot“-Attribut. Folglich umfasst die RODL im „Worst-Case“, mit 16 Runden à 255 „Slots“, 16·255·2 Bytes = 8160 Bytes. Diese Implementierungsart ist zwar nicht speicherschonend, dafür ist sie schnell, universell und ermöglicht eine Konfiguration zur Laufzeit. Falls die Geschwindigkeit nicht im Vordergrund steht, kann man die Bits der „Slot“-Attribute in einem ersten Schritt gepackt speichern. Dadurch erhöht sich der Verwaltungsaufwand und damit die Rechenarbeit, der Speicherbedarf nimmt im „Worst-Case“ jedoch um ca. 44 % ab (16·255·(1 Bytes + 1 Bit) = 4590 Bytes). Im zweiten Schritt besteht die Möglichkeit die RODL als verkettete Liste zu organisieren, vorausgesetzt der „Master“ nimmt nur an ein paar „Slots“ teil und soll andere „Slots“ nicht mithören. Das reduziert wiederum den Speicherbedarf und erhöht die Rechenarbeit. Weitere Beispiele hierzu finden sich in Abschnitt 3.2.3. Es gibt also nicht die optimale RODL-Struktur, sondern eine, bezüglich Ressourcen, Geschwindigkeit, Universalität und Flexibilität, angepasste RODL-Struktur – das ist für einen „Master“ und vor allem für die ressourcenärmeren „Slaves“ wichtig. Mit der RODL-Struktur ändert sich natürlich auch das eine oder andere Implementierungsdetail, allerdings nicht das Prinzip.

Aus diesem Grund kann der „C16x-Master“ nicht allen Anforderungen gerecht werden. Er wurde für Forschungszwecke entworfen und ist dementsprechend universell und flexibel einsetzbar. Trotzdem eignet sich der „C16x-Master“ für Aussagen über spezialisierte und optimierte Implementierungen

[117] Ein C16x-µC hat mit f_{CPU} = 20 MHz eine max. Rechenleistung von 10 MIPS.

oder Portierungen. Hinsichtlich des Speicherbedarfs fallen sie nämlich allesamt kleiner aus. Je nach Spezialisierungs- und Optimierungsgrad, wird sich die Codegröße zwischen 1,2..1,5 KB bewegen. Bei den Daten spielt zudem die Anwendung eine Rolle. Mit wenigen „Slots“ wird ein „Basic-TTP/A-Master“ nicht mehr 100 Datenbytes benötigen, wohingegen mit vielen „Slots“ auch 1 KB und mehr möglich sind.

5.2.6 „C16x-Slave“

Baudrate und Rechenlast

Der „C16x-Slave“ („Basic-TTP/A-Slave“-Implementierung plus C16x-Portierung) ist dem „C16x-Master“ sehr ähnlich. Er arbeitet ebenfalls nach den Prinzipien aus Kapitel 4, ist schnell, universell einsetzbar und zur Laufzeit umkonfigurierbar. Aufgrund dieser Eigenschaften, konnten mit dem „C16x-Slave“ sämtliche Versuche durchgeführt und alle Anwendungsbeispiele realisiert werden. Des Weiteren verwendet der „C16x-Slave“ die „Slot-Timer“-Lagefehlerkorrektur (siehe Abschnitt 4.1.2), wegen seines Quarzoszillators aber nicht die optionale „Slot-Timer“-Gangfehlerkorrektur (siehe Abschnitt 4.1.3). Bezüglich der CPU-Auslastung hat der „C16x-Slave“ deutliche Vorteile gegenüber dem „C16x-Master“. Das zeigen die in Bild 5.6 oszillographierten Zeitverhältnisse bei den Baudraten 125 kBit/s und 625 kBit/s. Sie stellen die Pendants zu den Oszillogrammen des „C16x-Masters“ aus Bild 4.13 dar, wodurch ein direkter Vergleich möglich ist.

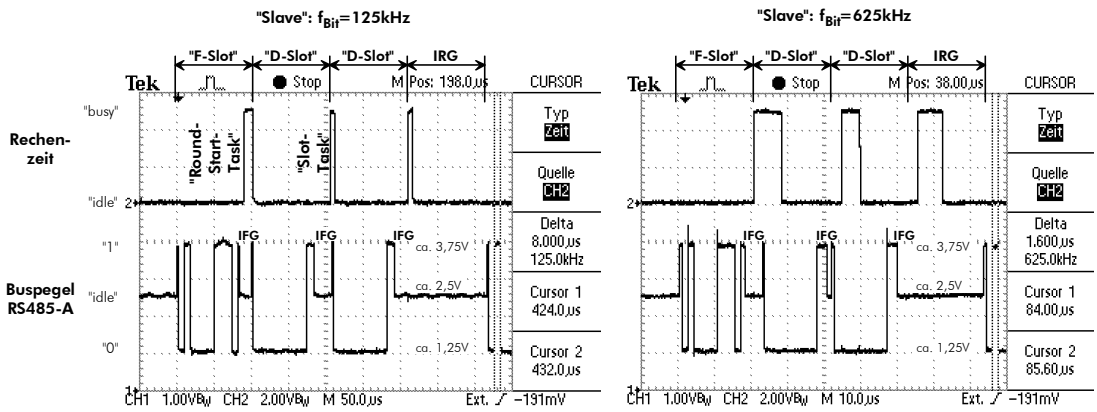


Bild 5.6: Beispiele für die Rechenzeiten eines „C16x-Slaves“

Als erstes fällt auf, dass der „C16x-Slave“ an allen Daten-„Slots“ teilnimmt. Er macht das nicht wirklich, obwohl er mit einem entsprechenden „Schedule“ dazu in der Lage wäre. Stattdessen prüft er im ersten Daten-„Slot“ nur die Teilnahme, während er am zweiten Daten-„Slot“ lesend teilnimmt. Man kann das im rechten Oszillogramm erkennen. Dort ist die „Slot-Task“ nach dem ersten Daten-„Slot“ etwas kürzer als nach dem zweiten Daten-„Slot“, was eine Folge des geringeren Rechenaufwands bei einer Prüfung ist. Zudem identifiziert die Absenz der „Transmit-Task“ den Lesevorgang im zweiten Daten-„Slot“. Der Vorteil dieser Implementierung ist ihre Universalität. Sie eignet sich gleichermaßen für „Slaves“ die wenige und viele „Slots“ belegen. Allerdings belastet sie die CPU stärker als eine spezialisierte Lösung. Deshalb haben die nachfolgenden quantitativen Aussagen zur Rechenlast den Charakter einer oberen Schranke.

Ein zweiter Punkt ist die große Rechenzeitpause zwischen dem Ende einer Runde und dem Beginn der nächsten Runde. Sie entsteht, weil ein „Slave“ weder eine Runde vorbereiten noch einleiten muss. Aus

diesem Grund dauert die „Slot-Task“ am Rundenende nicht so lang wie beim „Master“ und sie muss zu Beginn des „Fireworks-Slots“ auch nicht aktiviert werden. Weil diese Rechenzeitpause in jeder „Slave“-Implementierung vorkommt, ist sie für die Durchführung der „Slot-Timer“-Lagefehlerkorrektur und gegebenenfalls „Slot-Timer“-Gangfehlerkorrektur prädestiniert. Ferner lieferte die Vermessung der beiden Oszillogramme aus Bild 5.6 eine relative CPU-Auslastung von ca. 23,8 % bei 625 kBit/s und ca. 4,8 % bei 125 kBit/s (mit Instrumentierungs-„Overhead“). Vergleicht man diese Werte mit denen des „C16x-Masters“, so stellt man fest, dass der „C16x-Slave“ weniger als die halbe Rechenarbeit produziert. Zusammen mit den gleichmäßigeren Rechenzeiten, tritt beim „C16x-Slave“ deswegen eine Realzeitverletzung durch „Task“-Überlappung erst bei viel höheren Baudraten ein als beim „C16x-Master“. Waren es dort, unter der Annahme einer Optimierung, ca. 1 MBit/s (CPU-Last ≈ 85,8 %), sind es beim „C16x-Slave“ etwa 2 MBit/s (CPU-Last ≈ 76,2 %).

Trotzdem tendiert ein „Slave“ zu niedrigeren Baudraten als ein „Master“. Man muss davon ausgehen, dass ein „Slave“ auch den ersten Daten-„Slot“ belegen können soll. Das gilt insbesondere für einen universellen „Slave“, wie dem „C16x-Slave“. In diesem Fall stellt die Reaktionszeit nach dem Empfang eines „Fireworks-Frames“ den kritischen Punkt dar. Sie wird durch die Geschwindigkeit des verwendeten µCs bestimmt, ist nahezu konstant („low-cost“ µCs) und muss kleiner als die IFG sein. Mit zunehmender Baudrate schrumpft jedoch die IFG, so dass früher oder später das Reaktionsvermögen des µCs ausgeschöpft ist. Ab diesem Punkt wächst der „Slot-Timer“-Lagefehler mit jeder weiteren Baudratenerhöhung an, weil sein Korrekturalgorithmus keine negativen Zeiten einstellen kann (siehe Abschnitt 4.1.2). Der „C16x-Slave“ erreicht diesen Punkt zwischen den Baudraten 312,5 kBit/s und 625 kBit/s. Eine genauere Bestimmung ist auf direktem Weg nicht möglich, weil der C16x-µC (f_{CPU} = 20 MHz) keine feinere Einstellung zulässt. Mittels der Messung aus Bild 4.25 (siehe Abschnitt 4.2.6) ist aber zumindest eine indirekte Abschätzung möglich. Es wurde die Wirkung der „Slot-Timer“-Lagefehlerkorrektur in einem „C16x-Slave“ oszillographiert und eine Dauer von ca. 5,6 µs für den Rundenstart festgestellt. Hieraus lässt sich in guter Näherung die „kritische“ Baudrate 2 Bit/5,6 µs ≈ 360 kBit/s berechnen.

Mit einer Optimierung und Spezialisierung, die aus Ressourcengründen für einen „Slave“ wichtiger sind als für einen „Master“, steigt diese Baudrate sicherlich bis auf 400 kBit/s an. Folglich weist der „C16x-Slave“ eine rechenleistungsbezogene max. Baudrate von ungefähr 40 kBaud/MIPS auf. In Analogie zum „C16x-Master“, eignet sich dieser Wert zur Abschätzung von Baudratenobergrenzen in zukünftigen „Slave“-Portierungen und -Implementierungen. Über die CPU-Auslastung muss man sich dabei keine bzw. kaum Gedanken machen, da ein „Slave“ vergleichsweise wenig Rechenzeit braucht; beim „C16x-Slave“ mit 400 kBit/s würde die CPU-Last lediglich 23,8 %·400/625 ≈ 15,2 % betragen. Angesichts dieser Erkenntnis stellt sich die Frage, warum der „C16x-Slave“ bei einer Baudrate von 625 kBit/s funktioniert. Es liegt an den sonstigen Zeitfehlern des „C16x-Slaves“. Wegen seines Quarzoszillators, liegen sie weit unter ihren Grenzwerten (siehe Kapitel 4), wodurch der „Slot-Timer“-Lagefehler größer ausfallen darf. Doch wie geschildert, ist das eine Sache der Rechenleistung und nicht etwa ein „Jitter“- oder anderweitiges Problem. Solange nämlich die „Slot-Timer“-Lagefehlerkorrektur arbeitet, wird der Restfehler nicht größer als ungefähr 500 ns (siehe Abschnitt 5.2.5). Das ist zwar relativ groß und liegt hauptsächlich an der vierstufigen „Pipeline“ des C16x-µCs, stellt für den „C16x-Slave“ aber kein Problem dar – „low-cost“ µCs haben meistens eine zweistufige „Pipeline“, weshalb ihr Einfluss i.A. geringer ist. Nach Abschnitt 4.1.2 gilt für den Restfehler $T_{0,\,Rest} \leq 0{,}3 \cdot T_{Bit,\,M}$, wodurch eine Restriktion erst ab ca. 0,3/500 ns = 600 kBit/s entsteht.

Speicherbedarf

Der Speicherbedarf des „C16x-Slaves“ entspricht in etwa dem des „C16x-Masters“. Auf der Codeseite werden etwas weniger als 1,5 KB benötigt und auf der Datenseite, je nach RODL-Größe, 100, 1000

oder mehr Bytes. Da ein „Slave“ im Allgemeinen spezialisiert ist, kann man seinen Speicherbedarf deutlich reduzieren. Zusammen mit einer Optimierung, ist eine Codegröße von ca. 1 KB wahrscheinlich (in einfachen „Slaves“ z.B. ist die Synchronisations- und „Error-Task“ oft unnötig). Bei den Daten spielt dagegen wieder die Anzahl der „Slots“ ein Rolle, an denen ein „Slave“ teilnehmen soll. Im Gegensatz zum „Master“, fällt sie in der Regel jedoch viel geringer aus. Ein typischer Sensor- oder Aktor-„Slave“ benötigt für den Prozessdatenaustausch ein und manchmal auch zwei „Slots“. Inklusive ein paar „Slots“ für z.B. Konfigurationszwecke und ein paar Bytes für sonstige Daten, dürfte sich der typische Datenspeicherbedarf zwischen 10..20 Bytes bewegen.

5.3 Einfaches Beispiel

Nachdem die wichtigsten Punkte der „Basic-TTP/A“-Implementierung erörtert sind, soll dessen Anwendung zuerst an einem einfachen und im nächsten Abschnitt an einem komplexen Beispiel demonstriert werden. Der Versuchsaufbau, der „Schedule“ und eine Übersichtskizze des einfachen Beispiels sind in Bild 5.7 zu sehen[118]. Es enthält ein kleines „Basic-TTP/A“-Netz mit einem Aktor-„Slave“, einem Sensor-„Slave“ und natürlich einem „Master“. Alle drei Knoten sind mit den „Target-Boards“ C165node II realisiert. Auf dem Aktor-„Slave“ befindet sich eine LED, die über den Bus steuerbar ist. Sie leuchtet, wenn mindestens einer der Taster auf dem Sensor-„Slave“ oder dem „Master“ gedrückt wird. Ansonsten ist die LED dunkel.

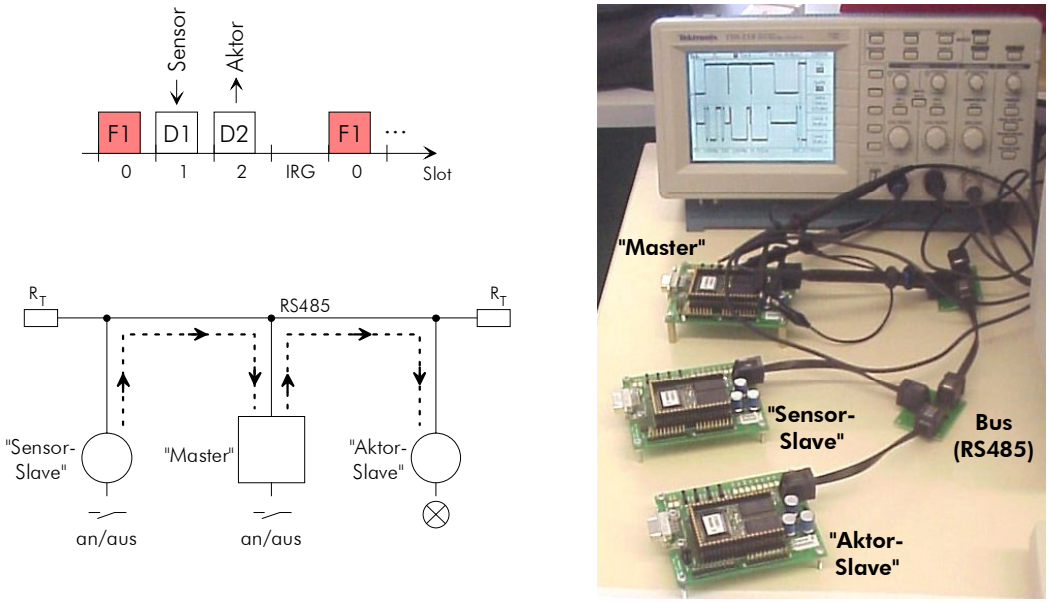


Bild 5.7 Einfaches Anwendungsbeispiel für „Basic-TTP/A“

Für diesen denkbar einfachen technischen Prozess gibt es viele unterschiedliche Ablaufpläne (Anzahl, „Schedules“ und Reihenfolge der Runden). Einer davon befindet sich in Bild 5.7 oben links. Er sieht eine drei „Slots“ lange, periodisch initiierte Runde vor. Die Rundennummer wurde willkürlich mit Eins festgelegt, weshalb der „Master“ in „Slot“ Null stets das „Fireworks-Frame“ F1 sendet. Anschließend überträgt der Sensor-„Slave“ in jedem ersten Daten-„Slot“ (D1) den Zustand seines Tasters. Gemäß dem gestrichelt eingezeichneten Informationsfluss, ist der „Master“ der Empfänger. Über

[118] Mit diesem Versuchsaufbau wurden die Oszillogramme in dieser Arbeit aufgenommen.

eine ODER-Verknüpfung mit seinem Tasterzustand, berechnet er daraus den Sollzustand der LED. Diese Information verschickt der „Master“ in jedem zweiten Daten-„Slot“ (D2) an den Aktor-„Slave“, wo sie ausgewertet und umgesetzt wird. Harte Realzeitrestriktionen existieren für diesen technischen Prozess nicht. Deshalb ist die Baudrate eine Nebensache und eine asynchrone Applikationsanbindung („Polling“ des entsprechenden Datums“) ausreichend.

Entsprechend einfach ist die Programmierung der Knoten. Sie soll am Beispiel der „Master“-Applikation in Bild 5.8 kurz erklärt werden. Da die Anwendung und die Schnittstellen von „Basic-TTP/A“ im Vordergrund stehen, entfallen Erläuterungen zur Umsetzung des ohnehin einfachen technischen Prozesses. In der ersten Codezeile wird der „Master“ mit `#include "ttp_master.h"` definiert. Seine Einrichtung und Initialisierung erfolgt in der Applikations-„Task“ `main()` über den Befehl `TTPa_Init(APPL_BAUD)`. Mit den anschließenden drei Befehlen wird die RODL gesetzt. Der Befehl `TTPa_SetLen(1,3)` sagt dem „Master“, dass die Runde eins existiert und drei „Slots“ lang ist. Als nächstes folgt die Beschreibung des ersten Daten-„Slots“ über den Befehl `TTPa_SetRodl(1,1,OFF,RD)`. Er ist als Lese-„Slot“ festgelegt („RD“), weil der „Master“ dort den Tasterzustand vom Sensor-„Slave“ erwartet. Die applikationsspezifische Konstante („Macro“) „OFF“ bestimmt den Startwert nach einem „Boot“-Vorgang. Analog dazu richtet der Befehl `TTPa_SetRodl(1,2,OFF,WR)` den zweiten und letzten Daten-„Slot“ der Runde eins als Schreib-„Slot“ („WR“) ein. Schließlich sendet der „Master“ in diesem „Slot“ die „ON/OFF“-Information an den Aktor-„Slave“. Der Startwert für diesen „Slot“ ist ebenfalls „OFF“, damit die LED nach dem „Booten“ nicht temporär flackert.

```
#include "ttpa_master.h"
#include "common.h"

void main(void)
{   UINT16 i;

    Board_Init();
    TTPa_Init(APPL_BAUD);
    TTPa_SetLen(1,3);
    TTPa_SetRodl(1,1,OFF,RD);
    TTPa_SetRodl(1,2,OFF,WR);
    TTPa_WriteRoundFifo(1);
    TTPa_Start();

    for(i=1;TRUE;i++)
    {    if (i >=  HEARTBEAT_TIME)
         {    Toggle(HEARTBEAT_SIGNAL);
              i = 1;
         }
         TTPa_WriteRoundFifo(1);
         if (KEY1 == KEYON || KEY == ON)
         {    LAMP = ON;
         }
         else
         {    LAMP = OFF;
         }
    }
}
```

```
void TTPa_ApplSyncHook(void)
{
}

void TTPa_ErrorHook(void)
{    ERROR_SIGNAL = ON;
     while (KEY2 == KEYOFF);
     ERROR_SIGNAL = OFF;
}
```

Bild 5.8: Quellcode der „Master“-Applikation

Mit dem nächsten Befehl `TTPa_WriteRoundFifo(1)` wird die erste Runde nach dem „Booten“ (hier die Nummer 1) festgelegt, womit der „Master“ startbereit ist. Der Start selbst geschieht über den Befehl `TTPa_Start()`. Anschließend folgt eine Endlosschleife, in der alle weiteren Befehle zyklisch und asynchron zu „Basic-TTP/A“ abgearbeitet werden. Hierzu gehören in erster Linie prozess-

spezifische Aufgaben, wie das Berechnen des Sollzustandes der LED für den Aktor-„Slave“. Daneben muss man im „Master“ noch die Rundensteuerung bedienen, weshalb sich der Befehl `TTPa_WriteRoundFifo(1)` ebenso in der Endlosschleife befindet. Eine ereignisabhängige oder synchronisierte Rundensteuerung ist in dieser einfachen Anwendung unnötig. Aus diesem Grund enthält die optionale, quasiparallele Synchronisations-„Task“ `TTPa_ApplSyncHook()`, die von „Basic-TTP/A“ in bestimmbaren „Slots“ aktiviert werden kann, keine Befehle. In der ebenfalls quasiparallelen „Basic-TTP/A-Task“ `TTPa_ErrorHook()`, befindet sich dagegen ein simpler „Error-Handler“. Wie geschildert, wird diese „Task“ bei einem Protokoll- oder Übertragungsfehler von „Basic-TTP/A“ angestoßen. Sie ermöglicht z.B. die Implementierung eines „Membership-Services“, weil „Basic-TTP/A“ auch leere „Slots“ erkennt, deren Ursache in der Regel ein Knotenausfall ist. Ein weiteres Beispiel für den „Error-Handler“ ist eine Extrapolation. Mit ihm können falsch übertragene Daten durch eine Modellrechnung ersetzt werden, sofern das in der Anwendung zulässig ist.

5.4 Komplexes Beispiel

5.4.1 Überblick

Das komplexe Beispiel soll zeigen wie vielfältig „Basic-TTP/A“ einsetzbar ist. Es handelt sich um das mehrfach erwähnte, in Bild 5.9 abgelichtete Demonetz mit Internet-Fernsteuerung. In der rechten Bildhälfte befindet sich das „Basic-TTP/A“-Netz mit den technischen Prozessen und in der linken Bildhälfte die Internet-Fernsteuerung. Das „Basic-TTP/A“-Netz besteht aus sechs „C165nodes I“, die auf dem vertikalen Brett montiert sind. Mit den oberen beiden „Slaves“ ist ein busgesteuerter Kfz-Blinker realisiert. In der Mitte befindet sich links ein „Stand-Alone-Master“ und rechts ein Bedienterminal-„Slave“. Die unteren beiden Knoten sind „Slaves“ in einem Regelkreis für einen Schwebekörper, der über den Bus geschlossenen ist. Als Busphysik wird der RS485-Standard mit ungeschirmten, nicht verdrillten Zweidrahtleitungen verwendet. Die Informationsübertragung findet mit einer Baudrate von 125 kBit/s statt. Eine der Besonderheiten des Demonetzes ist seine „Hot-Plug-In“-Fähigkeit. Sie gestattet das An- und Abstecken von Knoten während des Betriebs. Außerdem entkoppelt die Rückwirkungsfreiheit von „Basic-TTP/A“ die Applikationen, wodurch beim „Hot-Plug-In“ keine Störungen in unbetroffenen Anwendungen entstehen. So kann z.B. ein „Slave“ der Kfz-Blinkeranwendung ohne Beeinflussung der Schwebekörperregelung ausgetauscht werden.

Zu den insgesamt fünf „Slaves“ und dem „Stand-Alone-Master“ des „Basic-TTP/A“-Netzes, gesellen sich zwei weitere Knoten vom Typ „C166node“. Einer steht auf dem horizontalen Brett des „Basic-TTP/A“-Netzes unten links. Es ist ein simpler Busmonitor (TTP/A-Monitor), welcher den Datenverkehr des „Basic-TTP/A“-Netzes mithört, aufbereitet und am Bildschirm links von ihm anzeigt. Der zweite „C166node“ befindet sich auf dem „Tower“-PC ungefähr in der Mitte von Bild 5.9. Auf ihm läuft der „IFS/XML-Master“ für die Internet-Fernsteuerung. Je nach Versuch, muss das „Basic-TTP/A“-Netz entweder mit dem „Stand-Alone-“ oder dem „IFS/XML-Master“ betrieben werden. Die Funktion des „Web-Server“ übernimmt der „Tower“-PC und die des „Web-Clients“ das „Notebook“ auf dem Bildschirm. Mit Hilfe der „Web-Cam“ liefert der „Web-Server“, neben den standardmäßigen, textuellen Daten, eine kontinuierliche Bilderfolge über den technischen Prozess. Sie wird vom „Web-Client“ anzeigt und bietet auch bei großen Entfernungen eine visuelle Rückkopplung, wodurch sich der technische Prozess viel ergonomischer fernsteuern lässt. Das haben z.B. die Vorführungen auf der Messe Interkama 2001 sehr schön gezeigt, wo der „Web-Client“ auf der Messe in Düsseldorf stand und der Rest an der Technischen Universität in München.

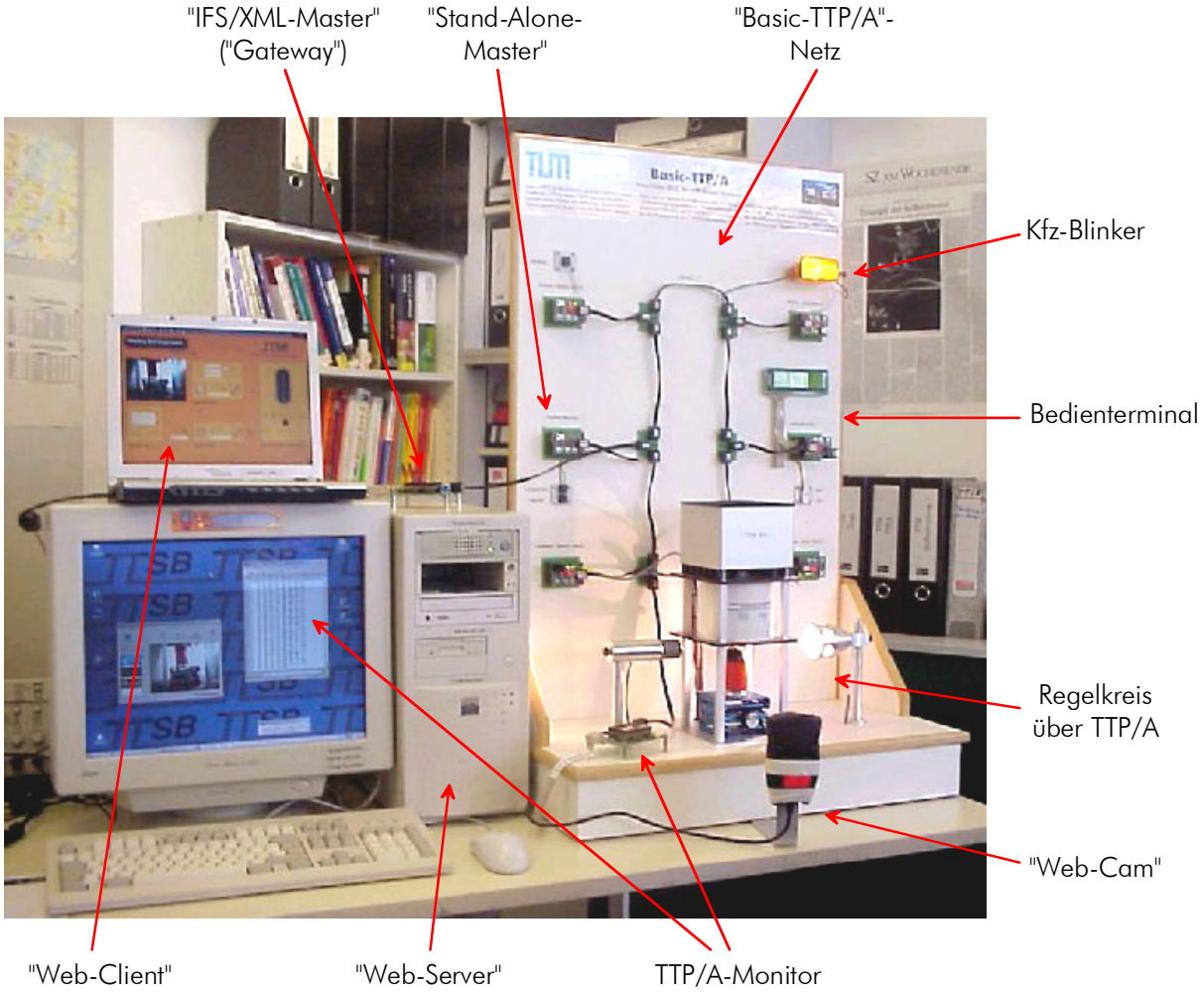


Bild 5.9: „Basic-TTP/A"-Demonetz mit Busmonitor und Internet-Fernsteuerung

5.4.2 „Basic-TTP/A"-Demonetz

Kfz-Blinker

Ein Beispiel für eine einfache Anwendung ist der busgesteuerte Kfz-Blinker. Der „Slave" mit dem Blinkertaster in Bild 5.9 oben Mitte ist ein binärer Sensor-„Slave". Er fragt den Zustand des Blinkertasters ab und macht aus den Impulsen ein bistabiles Ein/Aus-Signal. Mittels einer Runde fordert es der „Master" in regelmäßigen Abständen an und berechnet daraus ein Blinken/Aus-Signal. Dieses schickt er in ebenfalls regelmäßigen Abständen an den binären Aktor-„Slave" (Bild 5.9 oben rechts), der die Blinkerlampe über einen Schaltverstärker entsprechend ansteuert. Eine schematische Übersicht dieser und weiterer Zusammenhänge zeigt Bild 5.10. Da die Realzeitanforderungen an den busgesteuerten Kfz-Blinker unkritisch sind, laufen die Applikationen asynchron und sind ähnlich einfach, wie das Beispiel aus Abschnitt 5.3. Aus dem selben Grund müssen die Daten bzw. Signale nicht exakt periodisch übertragen werden, wodurch auch das „Scheduling" einfacher wird. Deshalb, und weil der „Master" die Hauptrechenarbeit übernimmt, stellt der busgesteuerte Kfz-Blinker ein Beispiel für den Einsatz von µCs der untersten Preiskategorie dar.

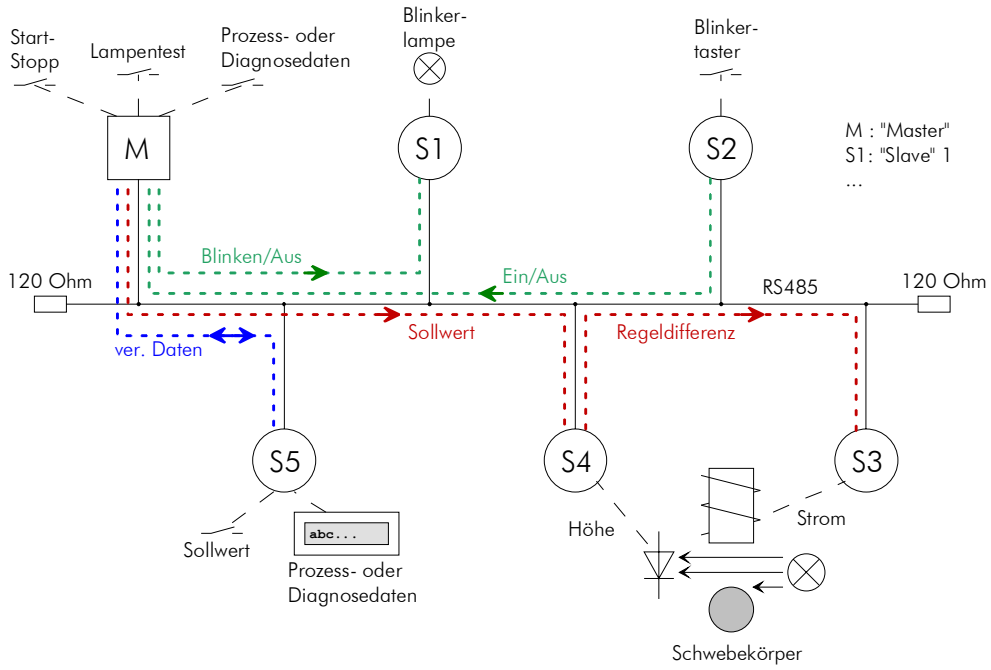


Bild 5.10: Schema des „Basic-TTP/A"-Demonetzes mit den wichtigsten Informationsflüssen

Regelkreis Schwebekörper

Im Gegensatz zum Kfz-Blinker ist die über den Bus geschlossene Abtastregelung für den Schwebekörper ein Beispiel für eine anspruchsvolle Anwendung. Das Funktionsprinzip lässt sich am besten aus Bild 5.10 erkennen. Mittels einer Lichtschranke misst der analoge Sensor-„Slave" S4 den Abstand zwischen dem ferromagnetischen Schwebekörper und dem Elektromagneten. Danach wandelt er den Messwert mit einem 12 Bit-A/D-Wandler in eine Ganzzahl um. Zur Unterdrückung von Störsignalen wird der gewandelte Messwert digital gefiltert. Aus diesem Istwert und dem, vom „Master" übermittelten, Sollwert berechnet der analoge Sensor-„Slave" S4 dann die Regeldifferenz. Unmittelbar darauf sendet er die Regeldifferenz an den analogen Aktor-„Slave" S3, auf dem ein parameteradaptiver, digitaler PD/PID-Ganzzahlregler mit Sättigungscharakteristik seine Dienste verrichtet. Aufgrund der direkten Punkt-zu-Punkt-Verbindung, ist die Totzeit minimal. Der analoge Aktor-„Slave" S3 ermittelt aus der Regeldifferenz eine Steuergröße, die er nach einer 12 Bit-D/A-Wandlung einem Stromverstärker zuführt. Sein Ausgangsstrom durchflutet den Elektromagneten so stark, dass die resultierende Magnetkraft den ferromagnetischen Körper in der Schwebe hält.

Die Abtastung des Regelkreises erfolgt zum einen äquidistant und zum anderen mit einer Frequenz von ca. 1 kHz. Bezogen auf das Gros der technischen Prozesse, sind das hohe Realzeitanforderungen. Deshalb ist das „Scheduling", auf das etwas später eingegangen wird, hier deutlich aufwändiger als beim Kfz-Blinker. Ferner werden die verteilten Algorithmen dieser Regelung mit den Mechanismen von „Basic-TTP/A" synchronisiert. Dazu sind die optionalen Synchronisations-„Tasks" in den beiden „Slaves" aktiviert und mit bestimmten „Slots" verknüpft worden. Auf diese Weise erfolgt ihre „Triggerung" durch den „Master", wodurch stets für den notwendigen Gleichtakt der entsprechenden, realzeitkritischen Algorithmen gesorgt ist. Weitere Details zum Regelkreis und zur Physik des Schwebekörpers sind im Anhang unter Punkt 8.12 nachzulesen.

Bedienterminal

Das Bedienterminal ist ein Anwendungsbeispiel für einen „low-cost“ Sensor/Aktor-„Slave“. Wie Bild 5.10 zeigt, findet zwischen dem „Master“ und dem entsprechenden „Slave“ S5 eine Kommunikation in beiden Richtungen statt. Als Sensoren fungieren zwei Taster und ein Schalter. Mit den Tastern kann man die Höhe des Schwebekörpers händisch verändern, währenddessen der Schalter eine automatische, sinusförmige Sollwertverstellung erlaubt. Hinsichtlich der Internet-Fernsteuerung, verschickt der auch genannte Terminal-„Slave“ S5 den jeweiligen Sollwert über den „Master“ an den Sensor-„Slave“ S4 des Regelkreises. Der Aktor des Terminal-„Slaves“ ist eine LCD-Anzeige zur Ausgabe von Prozess- und Diagnosedaten. Allerdings generiert der Terminal-„Slaves“ diese Daten nicht selbst, sondern er bekommt sie zeichenweise via „Basic-TTP/A“ vom „Master“. Das hat, neben der Flexibilität bei der Anzeige von Informationen, den Vorteil, dass man die erforderlichen Algorithmen und Daten auf den i.A. ressourcenstärkeren „Master“ verlagern kann. Insbesondere, wenn die komfortable, aber „speicherhungrige“ printf-Funktion verwendet wird, so wie hier geschehen.

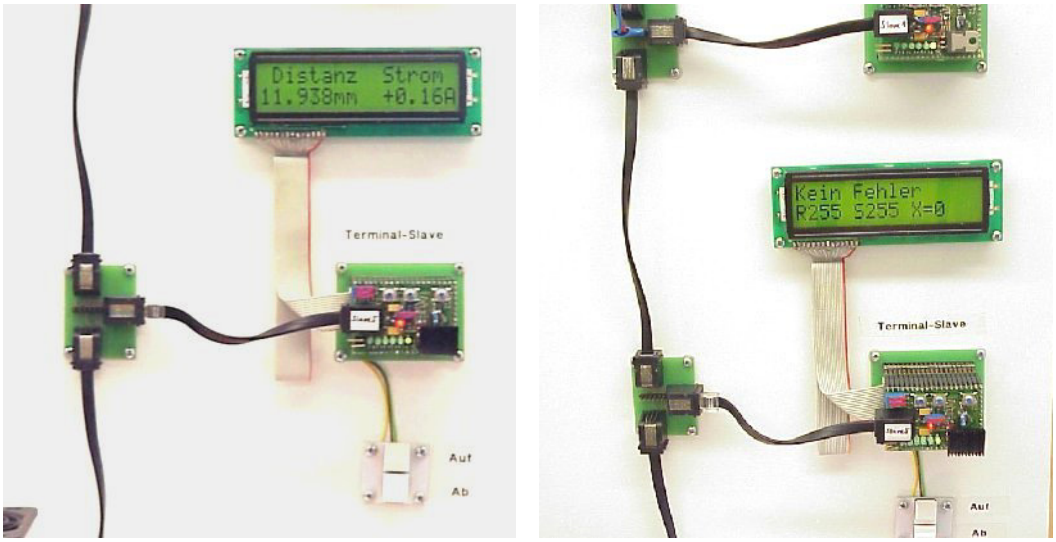


Bild 5.11: Bedienterminal mit einem Beispiel für Prozess- (links) und Diagnosedaten (rechts)

Zwei Beispiele für die Anzeige von Prozess- und Diagnosedaten zeigt Bild 5.11. In der linke Hälfte sind die Distanz und der Strom im Schwebekörperversuch zu sehen und der rechten Hälfte der Fehlerstatus von „Basic-TTP/A“. Da die Prozessdaten selbsterklärend sind und die Diagnosedaten nicht, werden nachfolgend nur Letztere erläutert. In der ersten Zeile der LCD-Anzeige steht die Art des zuletzt aufgetretenen Fehlers; „Basic-TTP/A“ erkennt immerhin acht verschiedene Fehler. Die zweite Zeile zeigt den Fehlerort (R: Runde, S: „Slot“) und die Anzahl der Fehler (X) seit dem letzten „Reset“ an. Mit Hilfe der „Schedules“ lässt sich damit ein ausgefallener oder störanfälliger Knoten lokalisieren, was z.B. für Wartungszwecke oder einen „Membership-Service“ von Bedeutung ist.

Ferner konnte eine interessante Erfahrung mit der „Basic-TTP/A“-typischen Idempotenz von Daten gesammelt werden. Aus Gründen des „Schedulings“, wird der Anzeigetext zeichenweise in den zyklischen Runden für den Kfz-Blinker- und Schwebekörperversuch übertragen. Seine Generierung hat eine niedrige Priorität, weshalb der „Master“ hie und da ein Zeichen verzögert sendet. Die normalerweise zuträgliche Datenidempotenz bewirkt nun, dass in diesen Lücken das zuletzt gesendete Zeichen mehrfach übertragen wird. Ohne Maßnahmen werden die betroffenen Zeichen entsprechend oft angezeigt, wodurch Fehler in den Texten und Werten entstehen. Deshalb wurde eine datenkonsumierende

Übertragung für die LCD-Anzeige entworfen, um eine Idempotenz der Zeichen zu verhindern – sonst ist das genau umgekehrt.

„Stand-Alone-Master"

Mit dem „Stand-Alone-Master" wird der isolierte Betrieb des „Basic-TTP/A"-Netzes demonstriert. Sein Pendant ist der „IFS/XML-Master", der im Wesentlichen zusätzlich die Funktion eines „Gateways" übernimmt. Auf ihn wird jedoch erst ihm nächsten Abschnitt eingegangen. Einige Aufgaben des „Stand-Alone-Masters" wurden bereits bei der Beschreibung der Anwendungen erklärt. Es handelt sich um die Steuerung des Kfz-Blinkers, die Zeitsteuerung der Schwebekörperregelung, die Sollwertabfrage vom Terminal-„Slave", die Auswertung von Protokollfehlern und deren Ausgabe auf dem Terminal-„Slave". Zu Letzterem sei ergänzend bemerkt, dass die Textausgabe aus Anwendersicht komfortabel und C-konform mit der printf-Funktion erfolgt. Alles Weitere übernimmt ein „Basic-TTP/A"-spezifischer „Low-Level"-Treiber, auf dem die printf-Funktion aufsetzt. Darüber hinaus besitzt der „Stand-Alone-Master" einen Taster für einen Lampentest und einen (Notaus)Schalter zum Stoppen der Applikationen. Mit diesen Beispielen wird zum einen die Realisierung von „Broadcasts" gezeigt und zum anderen, dass auch der „Stand-Alone-Master" Sensoren bzw. Aktoren bedienen kann. Außerdem ist der „Stand-Alone-Master" in der Lage, die Parameter des Reglers im analogen Aktor-„Slave" S3 zu verändern. Mittels eines weiteren Tasters ist damit eine PD/PID-Strukturumschaltung verwirklicht worden – eine Adaption gestattet die Internet-Fernsteuerung. Die Übertragung der Reglerparameter findet in einer sporadischen Runde statt. Erstens, um den ereignisgesteuerten „Master/Slave"-Anteil von „Basic-TTP/A" zu demonstrieren und zweitens, weil diese Informationen nur selten übertragen werden und keine Übertragungskapazität vergeuden sollen. Ein paar der aufgezählten Aufgaben werden synchron und die anderen asynchron bearbeitet. Zusammen mit den unterschiedlichen Schwierigkeitsgraden, deckt diese Applikation ein breites Anwendungsspektrum ab. Und schließlich verrichtet der „Stand-Alone-Master" die größte Rechenarbeit, womit diesbezüglich ein TTP/A-typischer Anwendungsfall aufgezeigt ist.

„Scheduling"

Prinzipiell würde eine lange Runde für das „Basic-TTP/A"-Demonetz ausreichen. Um jedoch die Flexibilität und Vielfalt von „Basic-TTP/A" unter Beweis zu stellen, sind die „Slots" auf drei Runden aufgeteilt. Die „Schedules" befinden sich in Bild 5.12, wobei angemerkt sei, dass es sich um eine und nicht um eine ausschließliche Lösung handelt. Als Rundennummern wurden die Eins, Zwei und Drei willkürlich festgelegt. Folglich belegen die „Fireworks-Frames" F1, F2 und F3 den jeweils ersten, farblich unterlegten „Slot" einer Runde. Des Weiteren sind die letzten beiden „Slots" in allen Runden schraffiert. In ihnen wird das einzige Datum mit harten Realzeitbedingungen, die Regeldifferenz, übertragen. Da die Regeldifferenz ein 12 Bit-Wert ist, benötigt sie zwei „Slots". Für die erforderliche Aufteilung wurde die UART-Regel „LSB first" angewandt, weshalb der vordere „Slot" das „Low"- und der hinteren „Slot" das „High"-Byte enthält. Konsequenterweise gilt diese Regel für sämtliche Mehrbytedaten; selbstverständlich darf die Aufteilung auch anderes und gemischt sein, solange sie konsistent ist. Der Grund für die Übertragung der Regeldifferenz in jeder Runde an der jeweils gleichen Stelle, liegt bei der geforderten äquidistanten Abtastung. Mit diesem Trick soll gezeigt werden, wie man derartige Realzeitbedingungen ohne Restriktionen für die Rundenreihenfolge erfüllen kann.

Die Positionierung der „Slots" für die Regeldifferenz ist auf folgende Überlegungen zurückzuführen. Für die A/D-Wandlung des Istwerts, dessen Filterung und die Berechnung der Regeldifferenz benötigt man etwas Zeit. Außerdem werden diese Vorgänge in der Synchronisations-„Task" von „Basic-TTP/A" erledigt, damit der verteilte Regelalgorithmus im Gleichtakt läuft – die „Triggerung" erfolgt in „Slave" 4 im „Slot" 6 und in „Slave" 3 im „Slot" 8. Beides zusammen disqualifiziert den ersten

Daten-„Slot“ in einer Runde für diesen Zweck. Alle anderen „Slots“ sind dafür geeignet. Dass die Wahl auf das Rundenende fiel, hat mit der prinzipiell abträglichen Totzeit in einem Regelkreis zu tun. In der IRG steht einer „Slave“-Applikation am meisten Rechenzeit zur Verfügung (siehe Bild 5.6 in Abschnitt 5.2.6). Folglich dauert die Berechnung eines neuen Stellwerts hier am kürzesten, wodurch die Totzeit minimal ist. Es handelt sich zwar um eine Nuance, weil das Gros der Totzeit durch die Übertragung entsteht. Dieses Beispiel soll jedoch vielmehr zeigen, dass man mit einem geschickten „Scheduling“ zumeist eine Optimierung ohne Aufwand erreicht.

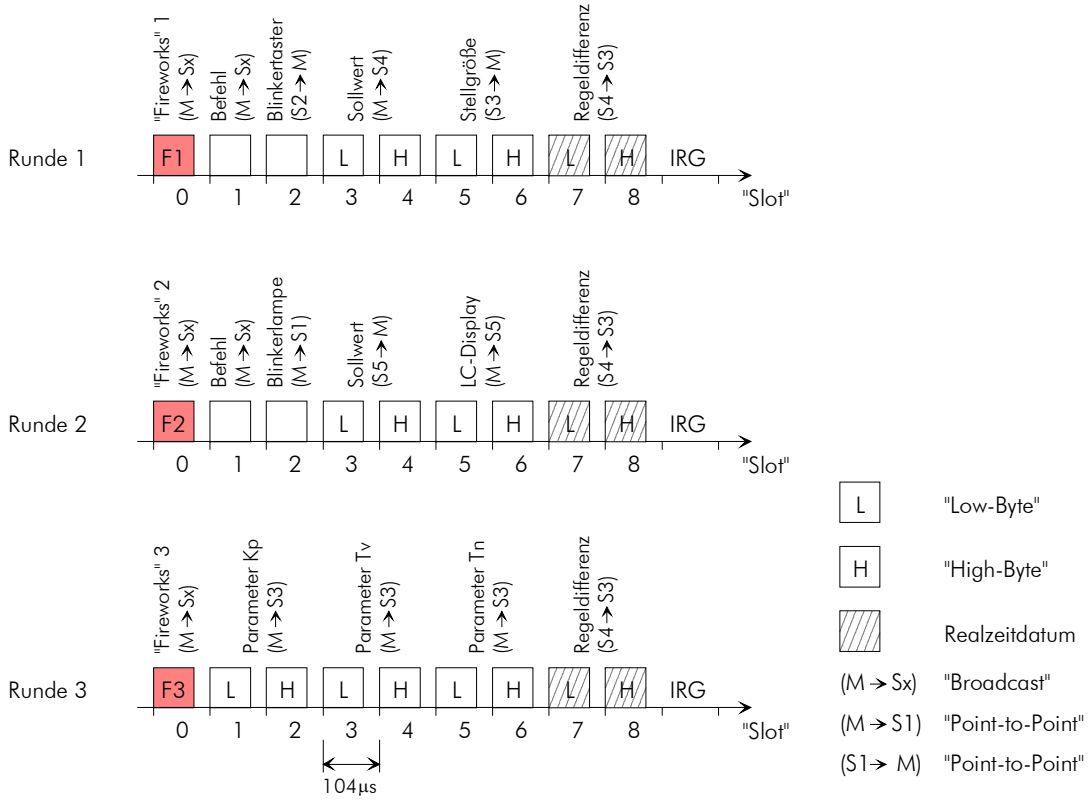


Bild 5.12: „Schedules“ des „Basic-TTP/A“-Demonetzes“

Für die Wahl der Rundenlängen und der Baudrate zeichnet die Abtastzeit der Regelung verantwortlich. Bei 125 kBit/s beträgt die „Slot“-Dauer 13·8 µs = 104 µs. Mit neun „Slots“ pro Runde, inklusive der IRG, ergibt das eine Rundenperiode von 1,04 ms. Aufgrund des „Scheduling“ und der gleichlangen Runden, fallen hier Runden- und Abtastperiode zusammen, womit die Forderung des Regelkreises erfüllt ist. Nun lässt sich auch dessen Totzeit abschätzen. Wie erwähnt, stößt „Slave“ S4 in jedem „Slot“ mit der Nummer Sechs die Synchronisations-„Task“ an, wodurch die aktuelle Regeldifferenz berechnet wird. Die Ausgabe des entsprechenden Steuerwerts geschieht, ebenfalls mittels der Synchronisations-„Task“ in „Slave“ 3, nach der Übertragung und Berechnung in der IRG. Weil die beiden Synchronisations-„Tasks“ nahezu an denselben relativen „Slot“-Positionen aufgerufen werden, resultiert eine ca. drei „Slots“ lange Verzögerung. Gemäß der „Slot“-Dauer von 104 µs, beläuft sich die Totzeit also ungefähr auf 300 µs.

In den restlichen „Slots“ befinden sich die weniger realzeitkritischen Daten. Unter Beachtung der Rundensteuerung ist ihre Belegung willkürlich. Die Runden eins und zwei werden abwechselnd und zyklisch initiiert, weshalb sie sämtliche Prozessdaten enthalten müssen. Zum ersten Daten-„Slot“ in diesen Runden gibt es anzumerken, dass in ihnen ein und dieselbe Information gesendet wird. Es ist

ein „Broadcast“, mit dem der „Master“ die Befehle für den Lampentest, den Notaus usw. versendet. Die doppelte Übertragung wurde in Kauf genommen, damit die „Slaves“ S1 und S2 der Kfz-Blinkeranwendung jeweils nur an einer Runde teilnehmen müssen. Als Folge benötigen sie weniger RODL-Speicher, was einer Realisierung mit „low-cost“ µCs, für das dieses Beispiel steht, entgegenkommt. Abschließend verbleibt die Runde drei, mit welcher der „Master“ die Reglerparameter in „Slave“ S3 ändern kann. Im Gegensatz zu den anderen beiden Runden, wird Runde drei bei Bedarf (Tastendruck) initiiert. Denn eine Änderung der Reglerparameter passiert aus Stabilitätsgründen relativ selten, wobei theoretisch kein Unterschied zwischen der realisierten PD/PID-Strukturumschaltung und einer stetigen Parameteradaption[119] existiert.

5.4.3 Erweiterungen

Busmonitor

Die erste Erweiterung des „Basic-TTP/A“-Netzes ist der anfangs erwähnte Busmonitor. Mit ihm lässt sich der Datenverkehr auf dem Bus leicht und übersichtlich beobachten. Ein Oszilloskop, so wie bei dem einfachen Beispiel aus Abschnitt 5.3, ist hier nicht mehr geeignet. Dafür sind erstens die Runden zu lang und zweitens der ständige Rundenwechsel zu komplex. Der Bus- bzw. TTP/A-Monitor wurde mit einfachen Mitteln und geringem Aufwand aufgebaut. Wie Bild 5.13 zeigt, besteht er aus einem C166node und einem Standard-PC.

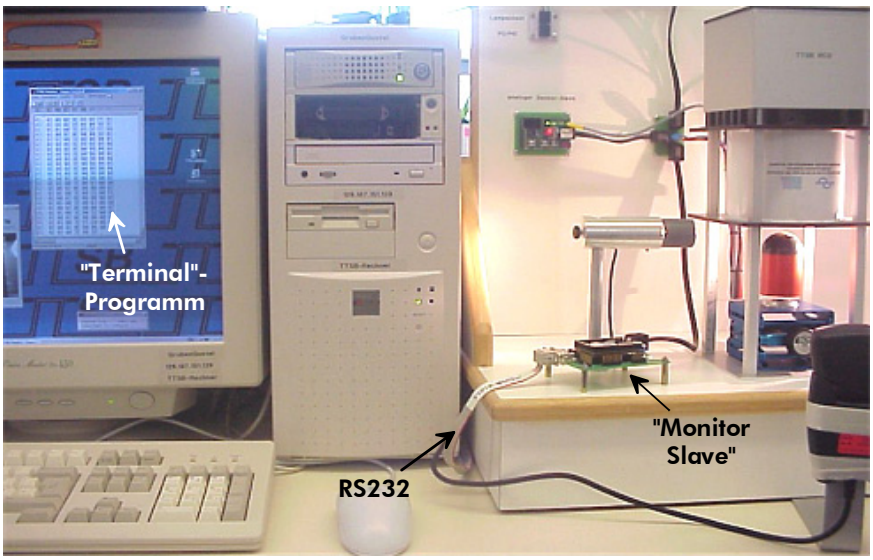


Bild 5.13: Einfacher Busmonitor für das „Basic-TTP/A“-Demonetz

Auf dem C166node läuft das eigentliche TTP/A-Monitor-Programm. Dank der universellen „Basic-TTP/A“-Implementierung, konnte es als passiver „Slave“ mit einer kleinen Applikation realisiert werden. Der „Slave“ ist so konfiguriert, dass er jeden „Slot“ mithört, protokolliert und auf alle Fehler reagiert. Auf diese Weise erhält man praktisch kostenlos die Trennung von Daten- und „Fireworks-Frames“, die Extraktion der darin enthaltenen Bytes und Informationen über die Art und den Ort von Fehlern. In der Applikation werden diese Daten benutzerfreundlich aufbereitet, nach Runden und „Slots“ geordnet und über die zweite serielle Schnittstelle des C166nodes (RS232) an den Standard-

[119] Aus Gründen des Stabilitätsnachweises sind parameteradaptive Regler i.A. quasistationär [137].

PC in der Mitte von Bild 5.13 gesendet. Dort läuft ein gewöhnliches „Terminal“-Programm (z.B. Hyperterminal oder Kermit), welches die Daten entgegennimmt, auf dem Bildschirm ausgibt und gegebenenfalls auf der Festplatte speichert. Natürlich lässt sich der TTP/A-Monitor hinsichtlich Funktionalität, Komfort und Bedienung verbessern. Aufgrund seiner Einfachheit, konnte jedoch zügig ein effektives Werkzeug geschaffen werden, das in den Analysen sehr hilfreich war.

Internet-Fernsteuerung

Die Internet-Fernsteuerung des „Basic-TTP/A“-Netzes ist die zweite Erweiterung. Sie ist in Bild 5.14 zu sehen und entstand aus einer Zusammenarbeit zwischen der Universität Stuttgart und der Technischen Universität München. Da die Idee von Universität Stuttgart stammt und sie den Hauptanteil der Arbeiten übernahm, wird die Internet-Fernsteuerung hier nur angerissen. Als erstes benötigt man ein „Gateway“ zum „Basic-TTP/A“-Demonetz (siehe Abschnitt 3.2.1). Es basiert auf „Basic-TTP/A“, im Sinne der TTP/A-Architektur (siehe Abschnitt 3.1). Somit stellt die „Gateway“-Erweiterung eine erste, wenn auch noch rudimentäre, Umsetzung des TTP/A-Bausteins IFS dar. Für die Realisierung wurde der „Stand-Alone-Master“ samt Applikation auf einen C166node portiert, weil dieses „Target-Board“ eine freie serielle Schnittstellen (RS232) besitzt. Anschließend erfolgte der Ausbau zum „Gateway“ mit dem Software-Baustein der Universität Stuttgart. Der Software-Baustein kommuniziert zum einen mit „Basic-TTP/A“ und zum anderen, über die freie Schnittstelle des C166nodes, mit einem Standardrechner. Aufgrund des Datenaustauschs mit dem Standardrechner im Internet-orientierten XML-Format, erhielt der C166node die Bezeichnung „IFS/XML-Master“.

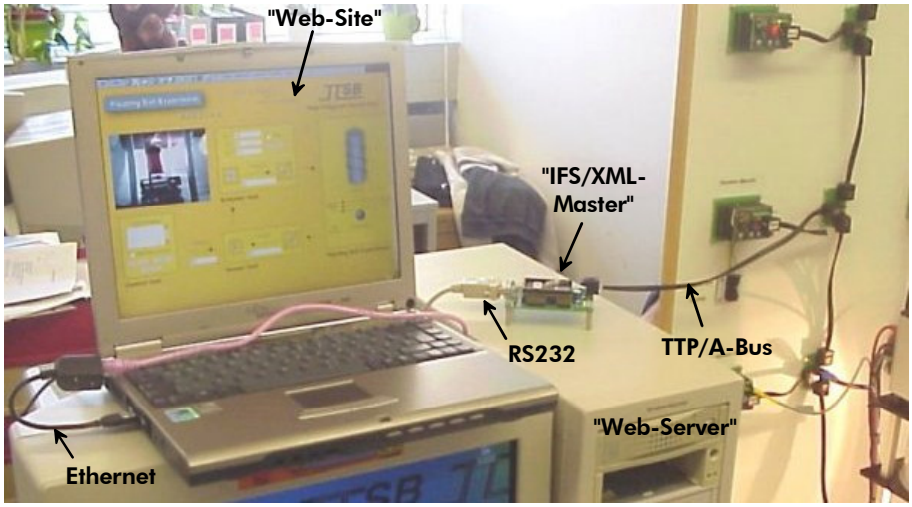


Bild 5.14: Internet-Fernsteuerung des „Basic-TTP/A“-Netzes

Auf dem Standardrechner, der in diesem Beispiel ein gewöhnlicher „Desktop“-PC ist, läuft ein „Web-Server“. Dank des XML-Ansatzes, muss dieser keine Datenkonvertierung o.Ä. vornehmen, weshalb der „Desktop“-PC weder TTP/A-spezifische Programme, noch TTP/A-spezifische Daten enthält. Das Einzige was der „Desktop“-PC speichert, ist eine „Web-Site“ für den Schwebekörperversuch. Sie wird über den „Web-Server“ via Internet für jedermann zugänglich gemacht[120]. Zur Fernsteuerung des Schwebekörperversuchs benötigt man lediglich irgendeinen Rechner mit Internet-Zugang und „Web-Browser“. Die „Web-Site“ enthält ein „Java-Script“, das beim Aufruf vom „Web-Browser“ ausgeführt wird. Es bewirkt, dass der „Web-Server“ mit dem „IFS/XML-Master“ zu kommunizieren beginnt.

[120] http://129.187.151.139/default_new.asp

Ferner sorgt das „Java-Script“ für den regelmäßigen Austausch der Prozessdaten zwischen der „Web-Site“ und dem „IFS/XML-Master“, wodurch sowohl deren Beobachtung als auch Manipulation möglich ist. Diese Situation ist in Bild 5.14 festgehalten. Auf dem „Notebook“ erkennt man die „Web-Site“ des Schwebekörperversuchs, über die sämtliche Regelkreisdaten zahlenmäßigen verfolgt werden können. Eine Manipulation wurde hingegen für den Sollwert und die Reglerparameter eingerichtet. Für eine visuelle Rückkopplung des Prozessgeschehens, zeigt die „Web-Site“ darüber hinaus einen „Video-Stream“ an. Nach dem Schließen der „Web-Site“ beendet der „Web-Server“ die Kommunikation mit dem „IFS/XML-Master“. Seitens der Universität Stuttgart ist geplant, diese Arbeiten in dem erwähnten Forschungsvorhaben „Smart-Transducer“ weiterzuführen. Unter anderem soll der „Web-Server“ auf den „IFS/XML-Master“ verlagert werden, damit ein direkter Internet-Anschluss möglich wird. Das hierfür notwendige „Target-Board“ mit Ethernet-Schnittstelle, der C165node IV, befindet sich gerade in der Entwicklung.

5.5 Zusammenfassung

Durch die erfolgreiche Implementierung von „Basic-TTP/A“ auf den mit Standardbauteilen aufgebauten C16Xnodes konnte beispielhaft gezeigt werden, dass sich das Konzept aus Kapitel 4 realisieren lässt. Es wurde Wert auf Universalität und Portabilität gelegt, weil die Implementierung momentan die Einzige ist. Sie unterstützt einen „Master“, einen „Slave“, jede RODL, ist in „C“ codiert und trennt Hardware-abhängige und -unabhängige Teile voneinander. Ferner enthält die Implementierung ein einfaches „Multitasking“-Schema, um auf Busereignisse und Fehler komfortabel reagieren zu können. Anhand von Anwendungsbeispielen hat die Implementierung ihre Funktionstüchtigkeit mehrfach bewiesen. Die gelungenen Arbeiten der Universität Stuttgart am IFS mit Internet-Anbindung und von SiemensVDO am „High-Speed-Physical-Layer“, die auf der „Basic-TTP/A“-Implementierung aufbauen, zeigen darüber hinaus die Vorteile der Skalierbarkeit und Offenheit von TTP/A, aufgrund der in Kapitel 3 vorgestellten Architektur.

Mit dem Aufbau eines Demonetzes wurde die Vielseitigkeit von „Basic-TTP/A“ vermittelt. Das Demonetz besteht aus einem busgesteuerten Kfz-Blinker als Beispiel für eine einfache Anwendung, einen über den Bus geschlossenen und gesteuerten Regelkreis als Beispiel für eine anspruchsvolle Anwendung, einem Bedienterminal als Beispiel für einen Multifunktions-„Slave“ und einem Busmonitor als Beispiel für die Umsetzung eines TTP/A-Werkzeugs und die Anbindung eines „Slaves“ an einen PC. Zudem konnten aus diesen und weiteren praktischen Erfahrungen erste Zahlenwerte über den Speicherbedarf und die baudratenbezogene Rechenleistung abgeschätzt werden: Ein „Master“ benötigt ca. 1,5 KB Codespeicher und je nach RODL-Größe ca. 100..1000 Bytes Datenspeicher. Für einen simplen „Slave“ genügen ca. 1 KB Codespeicher und ca. 10..20 Bytes Datenspeicher. Die maximale, rechenleistungsabhängige Baudrate liegt für beide Knotentypen bei etwa 50 kBaud/MIPS.

6 Redundanzkonzept

6.1 Allgemeines

Das Redundanzkonzept macht Vorschläge für den optionalen TTP/A-Baustein „Extended-Reliability“. Sie sollen im geplanten Forschungsvorhaben „Smart-Transducers“ umgesetzt, erprobt und gegebenenfalls erweitert werden (siehe Kapitel 7). Aus Abschnitt 3.4 ist bekannt, dass der Baustein „Extended-Reliability“ eine Erweiterung hinsichtlich des Einsatzes von TTP/A in sicheren bzw. verlässlichen Systemen darstellt. Voraussetzungen sind die Realzeitfähigkeit des Sensor/Aktorbusses, eine zuverlässige Datenübermittlung und eine angemessene Verfügbarkeit [103] [136]. Für die Realzeitfähigkeit und zum Teil auch für die zuverlässige Datenübertragung sorgt „Basic-TTP/A“. Den Rest übernimmt der Baustein „Extended-Reliability“. Er ist jedoch kein Garant für Sicherheit bzw. Verlässlichkeit, weil das gesamte System zählt und TTP/A stets nur ein Teil davon ist. Außerdem deckt der Baustein „Extended-Reliability“ nicht alle Anwendungsfälle ab. Dafür sind die Anforderungen schlicht zu vielseitig. In einem System mit Schwerpunkt Sicherheit ist die Fehlererkennung vorrangig (CRC, Knotenausfall...), wohingegen ein verfügbares System mehr Gewicht auf Fehlertoleranz legt (FEC, Ersatzknoten...). Deshalb und aus „Overhead“-Gründen sieht das Redundanzkonzept eine Skalierbarkeit des Bausteins „Extended-Reliability“ vor. Je nach Bedarf, verwendet man entweder alle standardmäßigen Erweiterungen oder nur die benötigten Teile. Auf diese Weise können, mit gegebenenfalls entsprechenden Ergänzungen, sämtliche Anwendungsfälle abgedeckt werden.

6.2 Schwachstellen und Randbedingungen

„Basic-TTP/A“ ist für die Kommunikation in einfachen, kostengünstigen Anwendungen konzipiert. Es erfüllt die Kriterien der Datenintegritätsklasse I1 nach DIN 19244 (zyklisch aufdatende Systeme) und besitzt so genannte „Single-Points-of-Failure“ bzw. „Golden-Units“. Für sichere bzw. verlässliche Systeme muss die Restfehlerwahrscheinlichkeit der Übertragung jedoch soweit abgesenkt werden, dass sie der Datenintegritätsklasse I3 (kritische Informationsübertragungen) genügt. Außerdem dürfen keine „Single-Points-of-Failure“ existieren, die Ausfallwahrscheinlichkeit wird anderweitig sehr gering gehalten oder es existiert ein „Fail-Safe“-Zustand.

Codierung

Die Reduzierung der Restfehlerwahrscheinlichkeit erfordert die Erweiterung der „Basic-TTP/A“-Codierung. Eine Änderung scheidet, wegen des Bausteinkonzepts von TTP/A, als Lösung aus. Ferner sollte die erweiterte Codierung möglichst wenig Ressourcen (Speicher und Rechenzeit) benötigen, um den Einsatz von „low-cost“ µCs weiterhin zu gewährleisten. Von den beiden Randbedingungen lässt sich die Erste problemlos einhalten. Die Zweite macht hingegen Schwierigkeiten, wenn man die Krite-

rien der Datenintegritätsklasse I3 vollständig erfüllen möchte. Bei einem gesicherten Telegramm hängen Rest- und Bitfehlerwahrscheinlichkeit typischerweise über eine Binomialverteilung zusammen[121]. Als Folge wächst die Restfehlerwahrscheinlichkeit in einem großen Bereich überproportional mit der Bitfehlerwahrscheinlichkeit an. Dieser Trend kehrt sich für Bitfehlerwahrscheinlichkeiten nahe der Eins um. Er führt dazu, dass die Restfehlerwahrscheinlichkeit zu Null wird, wenn die Bitfehlerwahrscheinlichkeit gleich eins ist – ein gesichertes Telegramm, das nur aus fehlerhaften Bits besteht wird i.A. immer erkannt. Die Probleme entstehen durch eine scharfe Restriktion der DIN 19244 in dem Bereich, wo die Restfehlerwahrscheinlichkeit überproportional mit der Bitfehlerwahrscheinlichkeit steigt (siehe Abschnitt 8.7 oder Bild 6.2). Bis zu einer Bitfehlerwahrscheinlichkeit von ca. $3 \cdot 10^{-4}$ nimmt die Grenzkurve für die Datenintegritätsklasse I3 ebenfalls überproportional mit der Bitfehlerwahrscheinlichkeit zu. Dann jedoch verharrt sie bei einem Wert von 10^{-12}, währenddessen die Restfehlerwahrscheinlichkeit weiterwächst.

Es Bedarf nicht viel sich vorzustellen, dass man für Bitfehlerwahrscheinlichkeit nahe der Eins, aber noch auf dem zunehmenden Ast der Restfehlerwahrscheinlichkeit, eine Hamming-Distanz von zehn oder größer benötigt, um die Restfehlerwahrscheinlichkeit unter die Grenzkurve zu drücken. Theoretisch ist das natürlich machbar, praktisch allerdings explodiert der Aufwand für die Codierung regelrecht. Die Folge ist ein übermäßiger Ressourcenbedarf, wodurch der Einsatz von „low-cost“ µCs gefährdet werden kann. Man benötigt erstens mehr Speicher und Rechenzeit für die Codierung und Decodierung und zweitens mehr Bandbreite und wiederum Rechenzeit, falls die Nettodatenrate gleich bleiben soll. Eine größere Bandbreite verringert jedoch den Signal-Rauschabstand, weswegen die Häufigkeit der Bitfehler in der Regel mit der Baudrate zunimmt. Darüber hinaus wachsen, aufgrund der längeren Codewörter, die Bitfehlermöglichkeiten. In Summe wird die Verbesserung der Restfehlerwahrscheinlichkeit also gedämpft und, nicht zu vergessen, es sinkt die Wahrscheinlichkeit für einwandfrei übertragene Codewörter. Aus diesen Gründen schlägt das Redundanzkonzept in Abschnitt 6.3 einen Kompromiss vor. Mit der erweiterten Codierung soll die Datenintegritätsklasse I3 unterhalb einer Bitfehlerwahrscheinlichkeit von 10^{-3} erreicht werden. Diese Randbedingung stellt praktisch keine Einschränkung dar, weil eine Bitfehlerwahrscheinlichkeit von 10^{-3} hoch ist [69] [100]. Zudem darf man bei sicheren bzw. verlässlichen Systemen eine gute elektromagnetische Verträglichkeit voraussetzen (differentielle Übertragung, verdrillte Leitungen, Schirmung, galvanische Trennung...).

Bezüglich des Fehlermodells kommt standardmäßig nur eine Extrapolation vom Grad 0 (Auslassung) in Frage. Zum einen ist in einem zeitgesteuerten Bus eine Wiederholung bei Bedarf nicht möglich [12] und zum anderen sorgt die typische Idempotenz von Daten für eine baldige, automatische Korrektur im Fehlerfall. Eine Extrapolation höheren Grades oder eine Vorwärtskorrektur (FEC) sind für den Baustein „Extended-Reliability“ zu speziell. Sie können aber relativ einfach hinzugefügt werden, da TTP/A skalierbar und offen ist und „Basic-TTP/A“ die notwendigen Voraussetzungen, insbesondere die priorisierte Fehlerbehandlung, bereithält.

Topologie

Um die Schwachstellen in der Topologie zu klären, wurde in Bild 6.1 ein „Basic-TTP/A“-Netz nebst Zuverlässigkeitsschaltbild skizziert. Es ist eindeutig zu erkennen, dass der „Master“ und der Übertragungskanal (kurz Kanal) Schwachstellen in „Basic-TTP/A“ darstellen. Wenn nur einer von beiden defekt ist, bricht die Kommunikation zusammen. Bei einem „Slave“ kann man davon nicht ausgehen, sofern es sich um keinen „Bubbling-Idiot“ handelt. Zur Vereinfachung der weiteren Betrachtungen

[121] Im Allgemeinen ist die Anzahl der Experimente die unabhängige Variable in der Binomialverteilung. Hier trifft das nicht zu, weshalb der gemeinte funktionelle Zusammenhang nicht mit der bekannten „Binomialkurve“ verwechselt werden darf.

wird dieser eher seltene Fall im Kanal berücksichtigt[122]. Aus diesem Grund ist die Annahme, dass ein „Slave“ keine zentrale Bedeutung hat gerechtfertigt. Folglich müssen die „Slaves“ im Zuverlässigkeitsschaltbild parallelgeschaltet werden. Die Parallelschaltung aller „Slaves“ wäre jedoch zu optimistisch, weshalb sie in Gruppen aufgeteilt sind. Üblicherweise steuert man mit einem „Basic-TTP/A“-Netz mehrere, unabhängige Anwendungen. Ein schönes Beispiel ist eine moderne Automobiltür mit Spiegelverstellung, Fensterheber, Zentralverriegelung und Licht. Der Ausfall des Aktor-„Slaves“ für den Fensterheber beeinträchtigt zwar die Anwendung selbst, wegen der Rückwirkungsfreiheit von „Basic-TTP/A“, jedoch nicht die Restlichen. Von einem Totalausfall kann man sprechen, wenn mindestens ein wichtiger „Slave“ in jeder Anwendung defekt ist. Deshalb sind diese allgemeinen Überlegungen sinnvoll und angebracht.

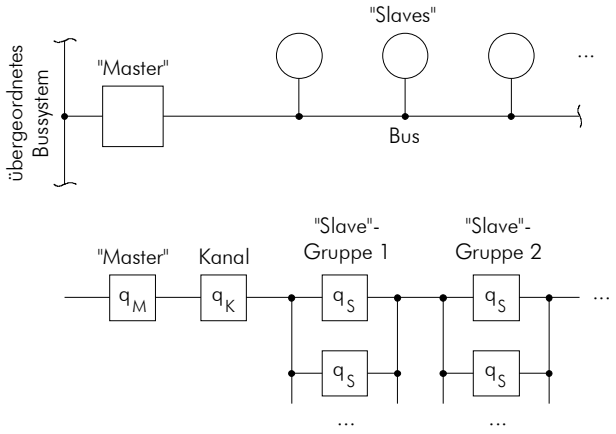


Bild 6.1: „Basic-TTP/A“-Netz (oben) und Zuverlässigkeitsschaltbild (unten)

Aufgrund der Parallelschaltungen, multiplizieren sich die Unverfügbarkeiten q_S der „Slaves“ einer Gruppe. Weiterhin ist die Unverfügbarkeit eines Bauteils bzw. einer Baugruppe i.A. sehr viel kleiner als eins (z.B. 10^{-3}). Demzufolge hat die Unverfügbarkeit einer Gruppe in Bild 6.1 deutlich geringere Werte als die eines einzelnen „Slaves“. Weil das in der Regel auch für den „Master“ und den Kanal gilt, dürfen die „Slaves“ in der Verfügbarkeit p_{Basic} bzw. Unverfügbarkeit q_{Basic} eines „Basic-TTP/A“-Netzes vernachlässigt werden. Dieser Sachverhalt ist in Formel 6.1 mathematisch beschrieben, wobei vereinfachend mit der gleichen Unverfügbarkeit q_S für alle „Slaves“ gerechnet wird und sich die Anzahl der „Slaves“ einer Gruppe im jeweiligen Exponenten von q_S (g_1, g_2...) äußert.

$$p_{Basic} = p_M \cdot p_K \cdot (1 - q_S^{g1}) \cdot (1 - q_S^{g2}) \cdot ...$$
$$q_{Basic} = 1 - p_{Basic}$$

$$q_M, q_K, q_S << 1, \text{ insbesondere } q_S^{g1}$$

$$\Rightarrow \quad p_{Basic} \approx p_M \cdot p_K$$
$$q_{Basic} \approx q_M + q_K$$

Formel 6.1: Verfügbarkeit p_{Basic} und Unverfügbarkeit q_{Basic} von „Basic-TTP/A“

[122] Daneben berücksichtigt der Kanal Kabelbrüche, Kurzschlüsse, „Stuck-Ats“ und Wackelkontakte (z.B. durch Stecker).

Damit ist gezeigt, dass in erster Linie die Unverfügbarkeiten des „Masters“ q_M und des Kanals q_K gesenkt werden müssen, um die Gesamtverfügbarkeit von „Basic-TTP/A“ zu erhöhen. Eine sehr wirksame, aber auch teure Methode ist die Replikation. Sie muss beim „Master“ angewendet werden, da es in TTP/A die Prämisse gibt nur Standardbauteile zu verwenden, insbesondere Standard-µCs. Außerdem würde ein, mit Spezialbauteilen aufgebauter, hochverfügbarer „Master“ wahrscheinlich mehr kosten, da die Stückzahlen von sicheren bzw. verlässlichen Systemen relativ niedrig sind. Beim Kanal dürfen auch konstruktive Maßnahmen, wie z.B. eine angemessene Kabelverlegung, Kabelqualität und Anschlusstechnik, angewandt werden. Sie sind meistens deutlich günstiger und weniger hochverfügbar als eine Kanalredundanz. Dass diese Methode keineswegs unsicher ist, zeigen die zertifizierten oder zur Zertifizierung anstehenden Bussysteme "Safety-at-Work“-Konzept (ASI), „SafetyBus“ (CAN), ProfiSafe-Bus, Interbus-Saftey, BST-Bus, Planet-Bus, TTCAN-Bus und einige mehr (siehe Kapitel 2). Ihre Anwendungsdomäne bilden kritische Systeme mit „Fail-Safe“-Zustand und hohem Kostendruck. Zwei typische Beispiele sind Lichtschutzgitter um Fertigungsroboter und „Airbag“-Steuerungen in Kfzs; bei einem Defekt bleibt der Fertigungsroboter sofort stehen und der „Airbag“ zündet nicht.

Trotzdem ist dieser Weg nicht immer probat. In „Fly-by-Wire“-, „Drive-by-Wire“-, Raumfahrt- und Zugsteuerungen reicht die Verfügbarkeit der genannten Bussysteme i.A. nicht aus. Dort werden kanalredundante Busse, wie z.B. der TTP/C-Bus, der FlexRay-Bus und der TCN-Bus, eingesetzt, weil es oftmals keinen „Fail-Safe“-Zustand gibt. Ein Flugzeug kann mit einer defekten Rudersteuerung genauso wenig sicher landen, wie ein PKW mit defekter Lenkung sicher stehen bleiben kann. Solche Systeme müssen entweder „fail-operational“ (sehr geringe Ausfallwahrscheinlichkeit) oder „fail-graceful“ (z.B. Geschwindigkeit reduzieren) sein [103] [136]. Oftmals wird eine hohe Fehlertoleranz auch vorgeschrieben und manchmal einfach nur gewünscht. Wie dem auch sei. Fakt ist, dass unterschiedliche Ausprägungen beim Begriff Sicherheit bzw. Verlässlichkeit existieren. Ein Beispiel liefert die Norm EN 954-1, indem sie fünf Kategorien für sicherheitsbezogene Teile einer Maschinensteuerung definiert (siehe Abschnitt 8.8). Folglich ist es unmöglich alle kritischen Anwendungen mit einer Methode optimal zu bedienen. Aus diesem Grund sieht das Redundanzkonzept für kritische Anwendungen mit hohem Kostendruck standardmäßig einen Schatten-„Master“ vor (siehe Abschnitt 6.4.1) – „Slaves“ können ohnehin repliziert werden – und für kritische Anwendungen mit sehr hohen Sicherheits- und Verfügbarkeitsansprüchen standardmäßig ein Parallelnetz (siehe Abschnitt 6.4.2)[123].

6.3 Erweiterte Codierung

6.3.1 „Fireworks-Frames“

Die „Fireworks-Frames“ besitzen durch die „Basic-TTP/A“-Codierung bereits eine gute Fehlersicherung. Wie in Abschnitt 3.2.8 und 4.3.3 gezeigt, beträgt ihre Hamming-Distanz vier und, aufgrund der wenigen Informationsbits, ihre Restfehlerwahrscheinlichkeit etwa $4 \cdot 10^{-12}$ für eine mittlere Bitfehlerwahrscheinlichkeit (kurz Bitfehlerwahrscheinlichkeit) von 10^{-3}. Dem zugehörigen Graph in Bild 6.2, am Ende dieses Unterpunkts, ist zu entnehmen, dass für Bitfehlerwahrscheinlichkeiten kleiner ca. $6 \cdot 10^{-4}$ sogar die Datenintegritätsklasse I3 nach DIN 19244 erreicht wird. Trotzdem muss die Redundanz der „Fireworks“-Codierung mit dem Baustein „Extended-Reliability“ erhöht werden. Denn es ist das erklärte Ziel, die Grenzkurve der Datenintegritätsklasse I3 spätestens bei der Bitfehlerwahrscheinlichkeit 10^{-3} zu unterschreiten. Leider ist im „Fireworks-Slot“, selbst für ein paar wenige zusätzliche

[123] Der FlexRay- und TCN-Bus können ebenfalls einkanalig betrieben werden. Beim FlexRay-Bus hat man dann die doppelte Bandbreite, wohingegen der zweite Kanal im TCN-Bus optional ist.

Prüfbits, kein Platz mehr vorhanden. Deshalb muss zur Erweiterung der „Fireworks“-Codierung der nächste „Slot“ belegt werden, wodurch ausreichend Platz für die zusätzlichen Prüfbits entsteht – inklusiv Paritybit neun Stück.

Theoretisch lässt sich mit den 18 Bits in den beiden „Frames“ ein (17,4)-Code[124] konstruieren. Ein (18,4)-Code ist nicht möglich, da die Paritybits aus den jeweiligen acht Informationsbits automatisch generiert werden. Um jedoch 17 Bits für die Codierung zu erreichen, muss man den Trick mit dem Paritybit aus Abschnitt 3.2.8 modifiziert anwenden. Jedes Paritybit erkennt alle ungeradzahligen Bitfehler in seinem „Frame“. Wenn ein ungeradzahliger Bitfehler in dem zusammengesetzten Codewort vorliegt, so bedingt das in jedem Fall einen ungeradzahligen Bitfehler in einem der beiden „Frames“. Folglich erkennen beide Paritybits alle ungeradzahligen Bitfehler – und auch ein paar geradzahlige Bitfehler – in dem zusammengesetzten Codewort. Nutzt man diese Eigenschaft zur Realisierung des üblichen Faktors (x + 1) in einem Generatorpolynom, so erhöht sich die effektive Bitstellenzahl um eins. Andernfalls wäre nur ein (16,4)-Code konstruierbar.

Eine rechnergestützte Analyse hat ergeben, dass die max. Hamming-Distanz für einen systematischen, linearen (17,4)-Code acht beträgt und, dass es 703 unterschiedliche (17,4)-Codes mit dieser Eigenschaft gibt. Ob sich darunter CRC-Codes befinden, wurde nicht untersucht. Aufgrund der Zerteilung des Codewortes und der beiden Paritybits, muss nämlich die „Burst“-Fehlererkennung eines CRC-Codes in Frage gestellt werden. Viel wichtiger ist die Kenntnis der maximal erzielbaren Hamming-Distanz mit einem systematischen, linearen Code, um die Effizienz und Güte der gewählten Methode bewerten zu können. Denn es macht aus folgenden Gründen keinen Sinn einen der 703 (17,4)-Codes im TTP/A-Baustein „Extended-Reliability“ zu verwenden: Die Suche nach einem (17,4)-Code, der alle „Fireworks“-Codes aus „Basic-TTP/A“ enthält, ist sehr aufwändig. Ferner ist die Wahrscheinlichkeit, dass ein solcher Code existiert, insbesondere wegen den beiden Paritybits, gering. Und selbst wenn, ist sein Ressourcenverbrauch höher, seine Implementierung umfangreicher, seine Ausführungszeit länger und vor allem seine Hamming-Distanz nicht größer als bei der gewählten Methode.

Gewählte Methode

Stattdessen sieht das Redundanzkonzept eine Wiederholung des „Fireworks“-Codes in „Slot“ 1 vor. Normalerweise ist diese Methode weder optimal, effizient noch elegant (siehe PPC-Bus in Abschnitt 2.3.4). Glücklicherweise trifft in diesem Fall das Gegenteil zu. Die Wiederholungsmethode ist abwärtskompatibel, wirkungsvoll, einfach zu realisieren, benötigt kaum Speicher und wenig Rechenzeit. Man muss lediglich in allen „Schedules“ die ersten Daten-„Slots“ im „Master“ schreibend und in den „Slaves“ lesend belegen. Des Weiteren ist ein simpler Vergleich nach dem Empfang beider „Frames“ erforderlich (auch im „Master“). Auf die CRC-Prüfung im Wiederholungs-„Frame“ kann dagegen verzichten werden. Sie passiert im Erfolgsfall indirekt und ist bei einem Fehler unnötig. Wenn „Basic-TTP/A“ im eigentlichen „Fireworks-Frame“ kein Fehler entdeckt hat und beide Codewörter gleich sind, also im Erfolgsfall, liefert jede weitere Auswertung dasselbe Ergebnis. Alle anderen Möglichkeiten decken sämtliche identifizierbaren Fehlerfälle ab. Da eine Vorwärtskorrektur beim „Fireworks-Code“, wegen der Fehlerfortpflanzungsgefahr, verboten ist, macht eine detaillierte Auswertung hier ebenso keinen Sinn. Aus dem gleichen Grund bleiben auch die Reaktionen nach dem Rundenbeginn dieselben. Im Erfolgsfall sieht ein Knoten die ermittelte Rundennummer als korrekt an und führt die entsprechende Runde aus. Im Fehlerfall verwirft er sie und wartet auf die nächste Runde.

Die wichtigste Eigenschaft aber ist die Verbesserung der Fehlererkennung. Hier bewirkt eine Wiederholung, dass sich die ursprüngliche Hamming-Distanz d_{min} verdoppelt. Wenn sie von Haus aus gering ist, fällt das Verhältnis „Δd_{min} zu redundante Bits“ schlecht aus. Deshalb hat die Wiederholungsme-

[124] 17 Bits insgesamt, davon vier für die Rundennummer.

thode keinen hohen Stellenwert in der Codierungstheorie. Bei den „Fireworks-Frames“ sind jedoch die Voraussetzungen anders. Sie haben durch „Basic-TTP/A“ bereits eine gute minimale Codedistanz. Aus diesem Grund führt ihre Wiederholung zu der beachtlichen Steigerung von $d_{min} = 4$ auf $d_{min} = 8$. Dass ein systematischer, linearer Code nicht mehr leistet, ist ein Zufall und lässt keine Verallgemeinerung zu. Man erreicht nämlich schon ab einem (15,4)-Code, also mit drei redundanten Bits weniger, die Hamming-Distanz $d_{min} = 8$. Da die „Frame“-Länge fest ist und die Wiederholungsmethode Vorteile bei der Realisierung hat, eignet sie sich hier aber besser.

Außerdem erkennt die Wiederholungsmethode zahlreiche Bitfehlerkombinationen, wodurch die Restfehlerwahrscheinlichkeit entsprechend klein ausfällt. Ein normales „Fireworks-Frames“ kann mit vier, sechs oder acht Bitfehlern unerkannt verfälscht werden. Allerdings müssen die Bitfehler in bestimmten Mustern vorliegen, weshalb jeweils nur ein paar Bitfehlerkombinationen dazu im Stande sind. Bei einem „Fireworks-Frame“ mit Wiederholung muss zusätzlich Gleichheit gelten, so dass ebenfalls nur wenige Acht-, Zwölf- und Sechzehnbitfehlerkombinationen unerkannt bleiben. Folglich lässt sich die Restfehlerwahrscheinlich für „Fireworks-Frames“ mit Wiederholung relativ einfach aus der Restfehlerwahrscheinlichkeit für normale „Fireworks-Frames“ ableiten – selbstverständlich mit demselben Kanalmodell (siehe Abschnitt 8.2).

$$P_{\text{Ext-Rest, Fireworks}} \approx 2^{-k} \cdot \sum_{e} \binom{9}{e} \cdot P_{\text{Bitfehler}}^{2 \cdot e} \cdot (1 - P_{\text{Bitfehler}})^{2 \cdot (9-e)} \qquad \text{für } e = 4, 6, 8 \quad \text{und } k = 5$$

Formel 6.2: Verbesserte Restfehlerwahrscheinlichkeit der „Fireworks-Frames“

Die einzigen Unterschiede in Formel 6.2 gegenüber Formel 4.16 befinden sich in den Exponenten der Eintritts- ($P_{Bitfehler}$) und der Nichteintrittswahrscheinlichkeit 1 - $P_{Bitfehler}$. Es ist jeweils der Faktor 2, mit dem die Paarung der Bitfehler berücksichtigt wird. Im Binomialkoeffizient und in der Reduktionsfunktion spiegelt sich hingegen die gleichgebliebene Anzahl der nicht identifizierbaren Bitfehlerkombinationen wider.

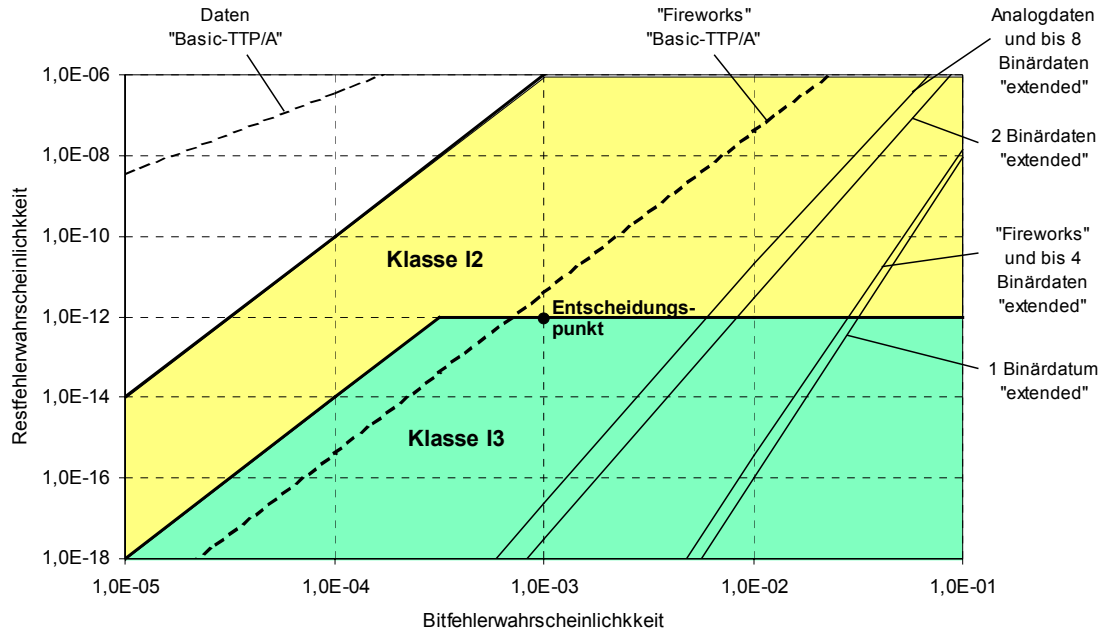


Bild 6.2: Sämtliche Restfehlerwahrscheinlichkeiten von TTP/A

Zur Beurteilung der Wiederholungsmethode wurde Formel 6.2 in Bild 6.2 graphisch ausgewertet. Darüber hinaus enthält dieses Bild die Verläufe der Restfehlerwahrscheinlichkeiten von „Basic-TTP/A“, die Verläufe der Restfehlerwahrscheinlichkeiten mit einer erweiterten Codierung für die Daten-„Frames“ (nächster Unterpunkt) und die relevanten Teile der Grenzkurven für die Datenintegritätsklassen I2 und I3 nach DIN 19244. Vorerst sind jedoch nur die Kurven für die „Fireworks-Frames“, in der Bildmitte und rechts, interessant. Sie belegen die deutlich verbesserte Fehlererkennung durch die Wiederholung der „Fireworks-Frames“. Mit „Basic-TTP/A“ erreicht die zugehörige Restfehlerwahrscheinlichkeitskurve das Gebiet der Datenintegritätsklasse I3 unterhalb einer Bitfehlerwahrscheinlichkeit von ca. $6 \cdot 10^{-4}$. Mit dem Baustein „Extended-Reliability“ passiert das Gleiche schon bei einer Bitfehlerwahrscheinlichkeit von etwa $4 \cdot 10^{-2}$. Als Ergebnis kann also festgehalten werden, dass das gesteckte Ziel, die Kriterien der Datenintegritätsklasse I3 unterhalb der Bitfehlerwahrscheinlichkeit 10^{-3} auf ressourcenschonende Weise zu erfüllen, mit der vorgestellten Methode für die „Fireworks-Frames“ erreicht wird.

6.3.2 Daten-„Frames“

Bei den Daten-„Frames“ ist die Basisfehlersicherung weit weniger gut als bei den „Fireworks-Frames“. Deshalb sind hier mehr zusätzliche Maßnahmen notwendig, um in das Gebiet der Datenintegritätsklasse I3 ab der Bitfehlerwahrscheinlichkeit 10^{-3} vorzudringen – es wird wieder mit dem Kanalmodell aus Abschnitt 8.2 gerechnet. Ein Problem bei den Daten-„Frames“ ist die unterschiedliche Länge der Informationen. Ein Daten-„Frame“ kann ein bis acht Informationsbits enthalten. Würde man mit nur einer Codierung arbeiten, die natürlich für acht Informationsbits ausgelegt sein müsste, so wäre die Übertragung bei Binärdaten ineffizient. Denn für eine konstante Restfehlerwahrscheinlichkeit steigen die Prüfbits überproportional mit den Informationsbits an. Binärdaten bzw. kleine Datenmengen sind aber für Sensor/Aktoranwendungen typisch (siehe Kapitel 1 und 2). Aus diesem Grund bietet das Redundanzkonzept eine stufenweise Codierung an, um die Übertragungseffizienz möglichst hoch zu halten. Dabei werden natürlich die atomare TTP/A-Dateneinheit, das Byte, und die geringen Ressourcen von „low-cost“ µCs berücksichtigt.

Binärdaten

Die erste Datenklasse bilden die häufigen Binärdaten. Im einfachsten Fall, bei einem Binär-„Slave“, ist die Information ein Bit lang. Ein „Slave“ kann jedoch auch mehrere binäre Sensoren oder Aktoren bedienen, wie das Beispiel Schalterbox für die Sitzverstellung in einem Fahrzeug zeigt (siehe Bild 2.5). Bei diesen so genannten Multibinär-„Slaves“ enthalten die Daten-„Frames“ mehrere Informationsbits. Alternativ könnte man jedes Binärdatum in einem separaten „Frame“ übertragen. Das wäre allerdings ungeschickt und ist unüblich, weil es den Aufwand erhöht, Ressourcen vergeudet und die Effizienz senkt. Aus diesem Grund wird dieser Fall außer Acht gelassen. Um zu klären wie viele Informationsbits man in einem Daten-„Frame“ adäquat sichern kann, wurden relevante (n,m)-Codes, unter Berücksichtigung des Paritybit-Tricks, rechnergestützt analysiert. Das Ergebnis ist in Tabelle 6.1 zu sehen:

Info-Bits	Prüfbits	d_{min}	$P_{Rest}(P_{Bit} = 10^{-3})$	Codebezeichnung
1	7	8	$\approx 1 \cdot 10^{-24}$	(8,1)-„Repetition“-Code
2	7	6	$\approx 3 \cdot 10^{-18}$	(9,2)-Code
3	6	4	$\approx 2 \cdot 10^{-12}$	(9,3)-Code ↯
4	5	4	$\approx 4 \cdot 10^{-12}$	(9,4)-„Fireworks“-Code ↯

Tabelle 6.1: Absicherungsgrenzen von Binärdaten in einem „Frame“

Gemäß Zeile drei, können nicht mehr als zwei binäre Signale pro Daten-„Frame“ angemessen gesichert werden. Darüber benötigt man einen zusätzlichen „Slot“, auch wenn die Restfehlerwahrscheinlichkeit für drei binäre Signale nur knapp über dem Entscheidungspunkt in Bild 6.2 ($P_{Rest.}(P_{Bit.} = 10^{-3}) \leq 10^{-12}$) liegt. Weil die erweiterte „Fireworks“-Codierung ebenfalls zwei „Slots“ braucht und eine sehr gute Fehlererkennung aufweist, bietet sie sich zur Absicherung von bis zu vier Informationsbits förmlich an. Zudem erfordert die erweiterte „Fireworks“-Codierung keine Ressourcen, weil im Baustein „Extended-Reliability“ bereits alles vorhanden ist. Etwas schwieriger bzw. aufwändiger sind mehr als vier Informationsbits zu handhaben. Noch einen Code zu entwickeln macht, hinsichtlich der ausstehenden Analogdatencodierung, kein bzw. wenig Sinn. Stattdessen können Multibinär-„Slaves“ mit 5..8 Sensoren und/oder Aktoren ihre Signale wie Analogdaten behandeln; Details folgen im Anschluss. Somit macht das Redundanzkonzept für die erweiterte Codierung von Binärdaten folgende Vorschläge:

Binär-signale	**„Slots“**	**Code**	**Codierung** M L P	**Alternative**
1	1	„Repetition“-Code[125]	0: 0000 0000 1 1: 1111 1111 1	erw. „Fireworks“-Codierung; Effizienzeinbuße 1 „Slot“
2	1	(9,2)-Code	00: 0000 0000 0 01: 0110 1101 1 10: 1011 0110 1 11: 1101 1011 0	erw. „Fireworks“-Codierung; Effizienzeinbuße 1 „Slots“
3..4	2	„Fireworks“-Code mit Wiederholung	siehe erweiterte „Fireworks“-Codierung (Abschnitt 6.3.1)	–
5..8	2	(17,8)-Code	siehe Analogdaten (nächster Punkt)	Aufteilung der Info-Bits; erw. „Fireworks“-Codierung; Effizienzeinbuße 2 „Slots“

Tabelle 6.2: Vorschläge zur Absicherung von Binärdaten

Beim „Repetiton“-Code kann man das Paritybit leider nicht nutzen. Schuld ist die gerade Anzahl der beeinflussbaren Bits in einem UART-„Frame“. Es ist leicht nachvollziehbar, dass die Summen der logischen Einsen entweder immer gerade oder ungerade sind. Folglich unterscheiden sich die Paritybits in den beiden Codewörtern nicht. Um das Gegenteil zu erreichen, müsste die Anzahl der beeinflussbaren Bits ungerade sein. Trotzdem genügen sieben Prüfbits für eine exzellente Fehlersicherung, wie der Graph ganz rechts in Bild 6.2 belegt. Anstelle der monotonen Bitmuster dürfen auch alternierende, zueinander inverse Bitmuster verwenden werden. Dieser Code heißt dann aber nicht mehr „Repetition“-Code. Im (9,2)-Code lässt sich das Paritybit wieder nutzen – von den zehn möglichen (9,2)-Codes wurde der in Tabelle 6.2 willkürlich ausgewählt. Es verbessert die Hamming-Distanz d_{min} um eins, gegenüber einem (8,2)-Code mit $d_{min} = 5$. Außerdem erkennt der (9,2)-Code alle ungeradzahligen Bitfehler, eben durch das Paritybit. Ein Vergleich mit dem „Repetition“-Code zeigt jedoch, dass die Hamming-Distanz des (9,2)-Codes deutlich geringer ist (siehe Tabelle 6.1). Aus diesem Grund hat die Restfehlerwahrscheinlichkeitskurve des (9,2)-Codes in Bild 6.2 einen relativ großen Abstand zu der des „Repetition“-Codes. Im Gesamten reichen die Fehlererkennungseigenschaften des (9,2)-Codes

[125] „Repetition“-Codes sind die einzigen binären Codes, welche die Singleton-Schranke erreichen. Die Singleton-Schranke bildet für lineare Codes ein Supremum bzg. des Codeabstands ($\max(d_{min}) = m + 1$, mit m: Anzahl der Prüfstellen) [104]. Außerdem erlauben „Repetition“-Codes eine einfache Vorwärtskorrektur, insbesondere ungeradzahlige „Repetition“-Codes.

aber vollends aus, da die Grenzkurve der Datenintegritätsklasse I3 weit vor dem Entscheidungspunkt durchstoßen wird.

Über die restlichen beiden Codes aus Tabelle 6.2 gibt es an dieser Stelle nicht viel zu berichten. Der „Fireworks-Code“ mit Wiederholung wurde bereits im vorherigen Abschnitt behandelt und auf den (17,8)-Code für Analogdaten und Binärdaten mit mehr als vier Bits wird später eingegangen. Daher soll nun diskutiert werden, ob vier Codierungen nicht zu viel sind. Natürlich gibt es Argumente dafür und dagegen. Zuerst das Wider: Aufgrund der unterschiedlichen Codes, ist kein einheitlicher Algorithmus vorhanden und es müssen Fallunterscheidungen getroffen werden. Das erschwert zum einen die Erstellung und Änderung von Applikationen und zum anderen wird der Baustein „Extended-Reliability“ umfangreicher und komplexer – das gilt nicht für den Laufzeitcode. Außerdem fallen die Restfehlerwahrscheinlichkeiten unterschiedlich aus (siehe Bild 6.2). Deswegen sieht das Redundanzkonzept eine Alternative vor. Sie ist in der letzten Spalte von Tabelle 6.2 eingetragen und verwendet ausschließlich den „Fireworks-Code“ mit Wiederholung. Auf diese Weise wird ein sehr guter, einheitlicher Schutz bei minimalen µC-Ressourcen geboten. Nachteilig ist jedoch die geringere Übertragungseffizienz. Sie wird, bis auf den Fall mit drei oder vier Informationsbits, um die Hälfte reduziert. Anstatt einem „Slot“ für ein bis zwei Binärsignale bzw. zwei „Slots“ für fünf bis acht Binärsignale, benötigt man bei der Alternative zwei bzw. vier „Slots“. Dadurch halbieren sich im Extremfall entweder die Reaktions- und Abtastzeiten oder die Baudrate steigt auf das Doppelte an. Sofern das für eine Anwendung oder die Knoten, vor allem für die „Slaves“, kein Problem darstellt, ist die Alternative anwendbar.

Für eine stufenweise Codierung sprechen dagegen folgende Argumente: Binäre „Slaves“ mit wenig Signalen werden in einem TTP/A-Netz häufig vorkommen. Erstens aus Kostengründen, zweitens aus Sicherheits- bzw. Verfügbarkeitsgründen und drittens aus Konzeptgründen. Es ist geplant in den „Slaves“ „low-cost“ µCs einzusetzen. Diese µCs haben, abzüglich von zwei bzw. drei I/O-Pins für einen Hard- bzw. Software-UART, oft nur ein oder zwei freie I/O-Pins. Beispiele sind die sehr günstigen 8-Pin-µCs der PIC12Cx-Familie von Microchip und der AVR-Familie von Atmel. Hinsichtlich der Sicherheit bzw. Verfügbarkeit ist ein Multibinär-„Slave“ deshalb ungünstig, weil durch seinen Ausfall mehrere Sensoren und/oder Aktoren gleichzeitig betroffen sind. Unter Umständen können dadurch die Beobachtbarkeit und Steuerbarkeit eines technischen Prozesses auf einen Schlag stark beeinträchtigt werden, wobei die Wahrscheinlichkeit mit der Anzahl der Sensoren und/oder Aktoren an einem Multibinär-„Slave“ steigt. Ferner spricht das Konzept der „Smart-Transducers“, also der intelligenten Sensoren und Aktoren, generell gegen den Einsatz von Multi-„Slaves“ mit vielen Sensoren und Aktoren. In einem „Smart-Transducer“ verschmelzen nämlich Sensor bzw. Aktor, µC, Busanschaltung und die restlichen Bauteile zu einer Einheit. Da viele Sensoren und Aktoren örtlich getrennt sind, kann man dieses Konzept bei Multi-„Slaves“ nicht oder nur selten anwenden. Eine der Ausnahmen ist die erwähnte Schalterbox zur Sitzverstellung in einem modernen Kfz.

Die Nichtfehlerwahrscheinlichkeit ist ein weiterer Aspekt. Je mehr Bits eine Nachricht besitzt, desto geringer ist die Wahrscheinlichkeit, dass die Nachricht ohne Fehler übertragen wird. Ein „Frame“ gelangt bei einer mittleren Bitfehlerwahrscheinlichkeit von 10^{-3} in 99,1 % der Fälle korrekt zum Empfänger. Für zwei „Frames“ beträgt die Nichtfehlerwahrscheinlichkeit bereits 98,2 %, für drei „Frames“ 97,3 % und für vier „Frames“ 96,4 %. Außerdem ist der Ressourcenbedarf des „Repetition“-, (9,2)- und „Fireworks“-Codes mit Wiederholung gering. Der „Repetition“-Code benötigt zwei, der (9,2)-Code vier Konstanten[126] und der „Fireworks“-Code ist bereits vorhanden. Beim (17,8)-Code hingegen ist der Ressourcenbedarf größer. Je nach Implementierung, wird entweder mehr Speicher oder mehr Rechenzeit gebraucht; Details folgen im Anschluss. Da Multibinär-„Slaves“ mit vielen Sensoren

[126] Konstanten befinden sich im Programmspeicher (FLASH, EPROM), wovon auch „low-cost“ µCs i.A. genug besitzen.

und/oder Aktoren aber ohnehin wenig Sinn machen, spielt dieser Punkt eine untergeordnete Rolle. Weiterhin ist der Baustein „Extended-Reliability“ skalierbar, so dass ein Knoten immer nur die benötigte Codierung speichert. Das gilt insbesondere für die „Slaves“, weil sie zumeist deduziert sind. Wieso sollte ein simpler Binär-„Slave“ den (9,2)-Code oder einen anderen speichern, wenn er mit dem „Repetition“-Code arbeitet? Beim „Master“ ist die Sachlage unterschiedlich. Er kommuniziert in der Regel mit allen „Slaves“ und muss deshalb oft alle Codes parat halten. Als ressourcenstärkster Knoten, hat der „Master“ damit aber kein Problem. Und schließlich ist die Übertragungseffizienz mit einer stufenweisen Codierung besser als mit einer monotonen Codierung. Summa summarum überwiegen zwar die Vorteile, dennoch soll hier keine endgültige Entscheidung getroffen werden. Zum einen sind auch Mischformen denkbar und zum anderen handelt es sich um Vorarbeiten für das geplante Forschungsvorhaben „Smart-Transducers“.

Analogdaten

Die andere Datenklasse bilden die Analogdaten. Unter Analogdaten sollen digitalisierte Analogwerte verstanden werden. Ihre Breite liegt üblicherweise zwischen acht und sechszehn Bits. Daraus folgt, dass inklusive einer guten Fehlersicherung für ihre Übertragung mindestens zwei UART-„Frames“ benötigt werden. Aufgrund der unvermeidlichen Codewortzerlegung, können „Burst“-Fehler in ihrem Erscheinungsbild variieren. Es besteht die Möglichkeit, dass zwei „Burst“-Fehler zu einem verschmelzen oder ein „Burst“-Fehler in zwei zerfällt. Das kann, muss aber nicht ihre Erkennbarkeit beeinflussen. Deshalb wird bei der erweiterten Codierung für Analogdaten kein übermäßiger Wert auf einen CRC-Code gelegt. Außerdem erreicht ein CRC-Code, wie sich zeigen wird, unter den noch festzulegenden Randbedingungen, nicht die max. Hamming-Distanz.

Die Randbedingungen werden durch die Kosten indirekt definiert. TTP/A-„Slaves“ müssen sehr günstig sein, weshalb vor allem „low-cost“ µCs für deren Realisierung in Frage kommen. „Low-cost“ µCs verfügen aber über wenig Ressourcen, mit denen der Baustein „Extended-Reliability“ zurechtkommen muss. Bei den Analogdaten hat man nun folgendes Dilemma: Der Coderaum wächst exponentiell mit der Länge der Codewörter (2^n; n: Anzahl der Bits). Aus diesem Grund kann eine schnelle, tabellenorientierte Implementierung der erweiterten Analogdatencodierung leicht die Speichergrenzen eines „low-cost“ µCs sprengen. Ein 8-Bit-Analogdatum benötigt eine Tabelle mit 256 Einträgen. Abhängig von der Applikation, ist dafür noch in zahlreichen „low-cost“ µCs Platz vorhanden. Ein 16-Bit-Analogdatum belegt hingegen eine Tabelle mit 65536 Einträgen, was jeden „low-cost“ µC überfordert. Das andere Extrem bildet eine algorithmische Lösung. Sie benötigt auch bei großen Analogdaten wenig Speicher. Allerdings dauert die Berechnung der Prüfbits erstens unterschiedlich und zweitens häufig sehr lang, so dass diese Lösung in vielen Fällen ausscheidet.

Folglich muss man einen Kompromiss zwischen dem Ressourcenbedarf, der Geschwindigkeit und, nicht zu vergessen, der Übertragungseffizienz schließen. Darin sollen Analogdaten byteweise übertragen und mit einem Extra-„Slot“ gesichert werden. Auf diese Weise bleibt der Coderaum überschaubar groß und begünstigt eine tabellenorientierte Realisierung. Denn gerade „low-cost“ µCs mit ADCs[127] besitzen oftmals mehr Speicher als ihre rein digitalen Pendants. Beispiele sind der PIC 16F73 von Microchip und der AT 90S4433 von Atmel mit je 4 KB FLASH-Speicher. Sollte der Speicher, wegen einer großen Applikation oder, weil von Haus aus weniger vorhanden ist, nicht ausreichen, bietet sich eine Mischung aus tabellen- und algorithmusbasierter Lösung an [110]. Zur Effizienz gibt es anzumerken, dass sie im Mittelfeld liegt. 8 Bit-Analogdaten werden mit einer hohen, 10 Bit- oder 12 Bit-Analogdaten dagegen mit einer mäßigen Effizienz übertragen. Der Grund ist die notwendige Aufteilung eines Datums auf mehrere „Slots“ ab mehr als acht Informationsbits. Während 8 Bit-Analogdaten mit

[127] DACs findet man auf µCs nicht. Stattdessen bieten sie die Möglichkeit einer DA-Wandlung über ein PWM-Signal.

der erweiterten Codierung in zwei „Slots“ übertragen werden, sind für 10 Bit- oder 12 Bit-Analogdaten vier „Slots“ erforderlich. Die Effizienzeinbuße kommt zu Stande, da 10 Bit- oder 12 Bit-Analogdaten die zusätzlichen „Slots“ gering auslasten. Bei 14 Bit- oder 16 Bit-Analogdaten ist die Auslastung und damit die Effizienz wieder besser. Letztere treten allerdings verhältnismäßig selten auf. Erstens benötigen nur wenig technische Prozesse diese feine Granularität und zweitens ist ein hoher schaltungstechnischer Aufwand nötig[128].

Mit zwei „Slots“ pro acht Informationsbits lässt sich ein linearer, systematischer (17,8)-Code konstruieren, sofern man den geschilderten modifizierten Paritybit-Trick anwendet. Dass sich dieser Trick auch hier lohnt, zeigte eine rechnergestützte Analyse mit dem erwähnten, selbstentwickelten Programm. Die zwei besten der 256 möglichen (16,8)-Codes weisen eine Hamming-Distanz $d_{min} = 5$ auf und die zwei besten der 512 möglichen (17,8)-Codes eine Hamming-Distanz $d_{min} = 6$. Zwischen den beiden (17,8)-Codes existiert kein wesentlicher Unterschied. Sie haben den gleichen Fehleraufdeckungsgrad und vom Aufbau her gleiche Generatorpolynome der Form $g(x) = g_1(x)\cdot(x + 1)$. Deswegen wurde es dem Zufall überlassen, einen dieser Codes für den Baustein „Extended-Reliability“ auszuwählen. Sein Generatorpolynom befindet sich in Formel 6.3:

$$g(x) = (x^8 + x^7 + x^6 + x^4 + x^2 + x + 1) \cdot (x + 1) = x^9 + x^6 + x^5 + x^4 + x^3 + 1$$

Formel 6.3: Generatorpolynom für den (17,8)-Code

Zu den Polynomen $g_1(x)$ gibt es anzumerken, dass sie in beiden Fällen weder primitiv noch in Primitivpolynome zerlegbar sind. Folglich handelt es sich bei den zwei (17,8)-Codes um keine CRC-Codes. Aus den geschilderten Gründen spielt dieser Punkt hier allerdings eine Nebenrolle. Viel wichtiger ist der Gewinn in der Hamming-Distanz durch den modifizierten Paritybit-Trick. Denn er verringert die Restfehlerwahrscheinlichkeit stärker als die Erkennung von „Burst“-Fehlern.

Für die Restfehlerwahrscheinlichkeit des (17,8)-Codes als Funktion der mittleren Bitfehlerwahrscheinlichkeit gilt annähernd Formel 6.4. Eine Beurteilung erfolgt am besten anhand von Bild 6.2 (S. 162), wo der Graph, nebst anderen Restfehlerwahrscheinlichkeitsgraphen und Grenzkurven, eingezeichnet ist. Man erkennt, dass die Restfehlerwahrscheinlichkeit des (17,8)-Codes klein genug ist, um das gesteckte Ziel zu erreichen. Es ist sogar ein deutlicher Sicherheitsabstand zum Entscheidungspunkt vorhanden, mit dem Approximationsfehler, insbesondere durch die Reduktionsfunktion 2^{-k} (k: Anzahl der Prüfbits) [108], abgefangen werden können. Die Wahrscheinlichkeit hierfür ist aber gering, weil Formel 6.4 eher pessimistisch schätzt. Nicht berücksichtigt wurden nämlich die geradzahligen Bitfehler, welche durch zwei ungeradzahlige Bitfehler in den beiden „Frames“ eines zusammengesetzten Codeworts entstehen (1 + 5, 3 + 3, 1 + 7, 3 + 5...) und, aufgrund der beiden Paritybits, erkannt werden. Deshalb sollte man die Restfehlerwahrscheinlichkeitskurve des (17,8)-Codes in Bild 6.2 als Obergrenze betrachten.

$$P_{\text{Rest, Ext-Data}} \approx 2^{-k} \cdot \sum_{e} \binom{17}{e} \cdot P_{\text{Bitfehler}}^{e} \cdot (1 - P_{\text{Bitfehler}})^{17-e} \quad \text{für } e = 6, 8, 10, 12, 14, 16 \text{ und } k = 9$$

Formel 6.4: Restfehlerwahrscheinlichkeit des (17,8)-Codes

[128] Ansonsten können die niederwertigen Bits durch Rauschen und/oder einer schlechten EMV unbrauchbar werden. Deshalb finden diese Wandlungen auch in gut entkoppelten, externen Bausteinen statt.

Optional können Analogdaten in Nibbles zerlegt und mit der erweiterten „Fireworks“-Codierung übertragen werden. Dadurch sinkt der Ressourcenbedarf auf ein Minimum, bei gleichzeitig niedrigerer Restfehlerwahrscheinlichkeit. Allerdings verschlechtert sich die Effizienz, weil man anstatt zwei, vier „Slots“ für acht Informationsbits benötigt. Mischformen sind ebenfalls möglich, vor allem für 10 Bit-Analogdaten. Es könnten z.B. ein Byte mit dem (17,8)-Code und die restlichen zwei Bits mit dem (9,2)-Code für Binärdaten gesichert werden. Auf diese Weise lässt sich, bei etwa gleicher Restfehlerwahrscheinlichkeit (siehe Bild 6.2), ein „Slot“ einsparen. Dagegen ist ein weiterer Code zur Optimierung nicht empfehlenswert. Erstens wäre der Gewinn nicht sonderlich hoch, zweitens würde die Übersichtlichkeit leiden, drittens gibt es, wie gerade gezeigt, andere Alternativen und viertens kommen Analogdaten in einem Sensor/Aktorbus seltener vor als Binärdaten. Die endgültige Entscheidung fällt jedoch im geplanten Forschungsvorhaben „Smart-Transducers“.

6.4 Erweiterte Topologie

6.4.1 Schatten-„Master“

Die erste standardmäßige, topologische Erweiterung von „Basic-TTP/A“ durch den Baustein „Extended-Reliability“ ist der Schatten-„Master“. Gemäß Abschnitt 6.2, soll sie hauptsächlich kritische Anwendungen mit hohem Kostendruck bedienen (z.B. „Airbag“-Zündung). Es muss jedoch vorgesetzt werden, dass die Unverfügbarkeit des „Masters“ deutlich größer ist als die des Kanals ($q_M >> q_K$). Andernfalls ist die Maßnahme ineffizient. Da der „Master“ im Allgemeinen die komplexeste Einheit, mit den meisten Bausteinen, Lötstellen, Kontakten etc., in einem TTP/A-Netz ist, wird die Voraussetzung in vielen Fällen erfüllt. Ferner darf man davon ausgehen, dass in einer kritischen Anwendung der Kanal ausreichend geschützt wird. Das gilt insbesondere für die Kabelverlegung, die Qualität des Kabels und die Qualität der Steckverbinder, sofern vorhanden.

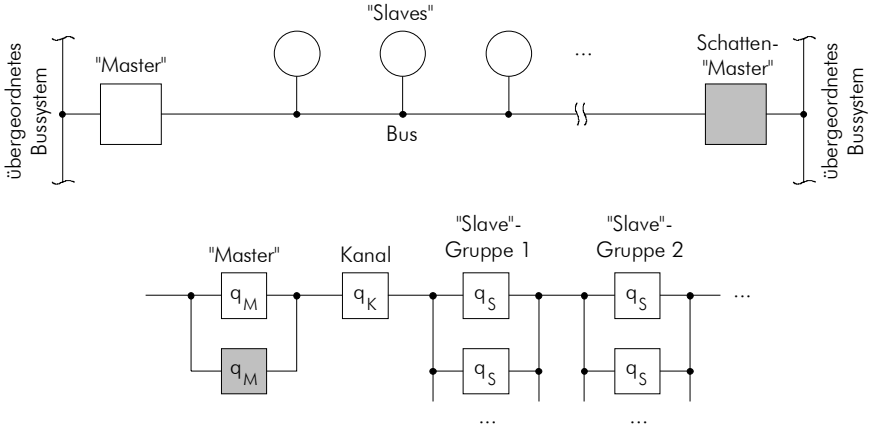


Bild 6.3: Topologie und Zuverlässigkeitsschaltbild eines „Basic-TTP/A“-Netzes mit Schatten-„Master“

Das Prinzip der Erweiterung durch einen Schatten-„Master“ ist einfach. Es ist in Bild 6.3, nebst Zuverlässigkeitsschaltbild, skizziert. An ein „Basic-TTP/A“-Netz wird ein nahezu gleich konfigurierter zweiter „Master“, der Schatten-„Master“, angeschlossen. Für die Koordination sorgt der Baustein „Extended-Reliability“. Er ist in der Regel auf allen Knoten und in Ausnahmefällen, wenn die standardmäßige, erweiterte Codierung nicht gewünscht wird, nur auf den „Mastern“ geladen. Der Busanschluss des Schatten-„Masters“ sollte am anderen Kabelende erfolgen. Auf diese Weise kann eine

Unterbrechung niemals beide „Master“ gleichzeitig vom Bus abtrennen. Zudem wären diversitäre „Master“ wünschenswert, falls es die Kosten gestattet.

Im Normalbetrieb verhält sich der Schatten-„Master“ ähnlich wie ein Busmonitor. Er beobachtet das Geschehen und sendet regelmäßig kurze Lebenszeichen an den „Master“ – dazu später mehr. Sobald der „Master“ ausfällt, übernimmt er die Buskontrolle und steuert entweder einen „Fail-Safe“-, „Fail-Graceful“-Zustand an oder hält den Betrieb aufrecht. Je nachdem, ob eine Anwendung mehr Wert auf Sicherheit oder Verfügbarkeit legt. Umgekehrt, wenn also der Schatten-„Master“ einen Defekt erleidet, passiert dasselbe. Außerdem besteht die Möglichkeit beide „Master“ an ein übergeordnetes Bussystem anzuschließen, z.B. mit der „Gateway“-Funktionalität des Bausteins IFS. Damit kann eine höhere Instanz (Leitebene) z.B. sämtliche Daten vergleichen und bei Unstimmigkeiten eine Ab-, Umschaltung o.Ä. einleiten. Ferner existiert die Möglichkeit das übergeordnete Bussystem als Brücke bei einem Kabelbruch zu verwenden, wodurch mit beiden „Mastern“ weiterhin alle „Slaves“ erreichbar bleiben. Ein Schatten-„Master“ kann also weitaus mehr sein als nur ein simpler „Master“-Ersatz. Für den Baustein „Extended-Reliability“ sind die genannten Beispiele allerdings zu speziell. Deshalb müssen solche Erweiterungen in der Applikation oder einem Anwenderbaustein erfolgen. Für partiell redundante „Slaves“, die sehr einfach hinzugefügt werden können, aber anwendungsspezifisch sind, oder z.B. Redundanz durch Sensorfusion gilt das ebenso.

Zur Beurteilung der Qualität und Effizienz der Maßnahme Schatten-„Master“, ist die Verfügbarkeit p_{SM} und die Unverfügbarkeit q_{SM} (Index SM: Schatten-„Master“) in Formel 6.5 berechnet. Es zeigt sich, dass die Verfügbarkeit p_{SM} bzw. die Unverfügbarkeit q_{SM} in guter Näherung jetzt nur noch durch die Verfügbarkeit p_K bzw. Unverfügbarkeit q_K des Kanals bestimmt wird. Im Gegensatz dazu, waren es bei „Basic-TTP/A“ die Verfügbarkeiten p_K und p_M bzw. Unverfügbarkeiten q_K und q_M des Kanals und „Masters“ (siehe Formel 6.1). Folglich wird mit dem Schatten-„Master“ einer der beiden „Single-Points-of-Failure“ beseitigt.

$$p_{SM} = (1 - q_M^2) \cdot p_K \cdot (1 - q_S^{g1}) \cdot (1 - q_S^{g2}) \cdot \ldots$$
$$q_{SM} = 1 - p_{SM}$$

$$q_M, q_K, q_S << 1, \text{ insbesondere } q_M^2 \text{ und } q_S^{g1}$$

$$\Rightarrow \quad p_{SM} \approx p_K$$
$$q_{SM} \approx q_K$$

Formel 6.5: Verfügbarkeit p_{SM} und Unverfügbarkeit q_{SM} mit einem Schatten-„Master“

Ein Zahlenbeispiel soll die Effizienz dieser Maßnahme belegen. Hierbei wird, wie eingangs geschildert, eine merklich höhere Unverfügbarkeit des „Masters“ im Vergleich zum Kanal angenommen ($q_M >> q_K$). Aus diesem Grund soll für das Beispiel gelten: $q_M = 10^{-2}$ und $q_K = 10^{-3}$. Gemäß Formel 6.1, hat ein „Basic-TTP/A“-Netz damit eine Unverfügbarkeit $q_{Basic} \approx q_M + q_K = 1{,}1 \cdot 10^{-2}$. Mit einem Schatten-„Master“ beläuft sich die Unverfügbarkeit lediglich auf $q_{SM} \approx q_K = 10^{-3}$, was einer relativen Abnahme von $(q_{SM} - q_{Basic})/q_{Basic} \approx -0{,}91$ entspricht – eine genauere Berechnung liefert einen Wert von etwa 0,9 und beweist, dass die Auswirkungen der vernachlässigten Terme marginal sind. Dieses Zahlenbeispiel zeigt, wie effizient diese Maßnahme sein kann, da die Ausfallwahrscheinlichkeit mit wenig Mehraufwand deutlich geringer ist. Allerdings gilt das nicht generell, sondern für den Fall $q_M >> q_K$. Der kommt zwar häufig, aber eben nicht immer vor. Werden die Rollen in dem Beispiel vertauscht,

also $q_M = 10^{-3}$ und $q_K = 10^{-2}$, so beträgt die relative Abnahme nur etwa -0,091, wodurch der Schatten-„Master“, zumindest für die Ausfallwahrscheinlichkeit, deutlich an Effizienz verliert. Für diese und weitere Fälle hält das Redundanzkonzept als Standardlösung das Parallelnetz unter Abschnitt 6.4.2 bereit.

Damit der Schatten-„Master“ dem „Master“ nicht in die Quere kommt, eine Ab- oder Umschaltung automatisch und rechtzeitig stattfindet, Alarme ausgelöst werden und ähnliche Dinge passieren, bedarf es einer Koordination. Diese Aufgabe übernimmt, wie erwähnt, der Baustein „Extended-Reliability“. Hierzu werden beide „Master“ nahezu gleich konfiguriert. Unterschiedlich ist lediglich die Definition des „Masters“ und Schatten-„Masters“ als solche, um klare Verhältnisse zu schaffen. Dadurch arbeitet der „Master“ wie gehabt, währenddessen der Schatten-„Master“ das Geschehen beobachtet und regelmäßig Lebenszeichen sendet. Außerdem vergleicht der Schatten-„Master“ seine Telegramme mit denen des „Masters“. Dazu arbeitet er die Runden genauso ab wie der „Master“ nur, dass seine Telegramme nicht am Bus erscheinen. Man kann das einfach realisieren, indem entweder der „Transmitter“ („Tristate“) oder der TxD-Pin des UARTs (Eingang) deaktiviert wird, je nach „Physical-Layer“. Auf diese Weise übernimmt „Basic-TTP/A“ den Vergleich, was sinnvoll ist, weil er dort ohnehin stattfindet. Sobald eine Ungleichheit auftritt, bekommt der Baustein „Extended-Reliability“ eine entsprechende Nachricht über die „Basic-TTP/A“-Fehlerschnittstelle, womit dem Bausteinkonzept von TTP/A Rechnung getragen ist. Weiterhin ermöglicht diese Technik eine schnelle Reaktion im Fehlerfall. Durch Aktivieren des „Transmitters“ oder TxD-Pins ist der Schatten-„Master“ unverzüglich in der Lage, die Buskontrolle zu übernehmen.

Für die notwendige Kommunikation zwischen dem „Master“ und dem Schatten-„Master“ wird, je nach Codierung, die Belegung der letzten zwei oder vier „Slots“ in allen Runden vorgeschlagen. Darin soll zuerst der „Master“ und dann der Schatten-„Master“ senden, um z.B. einen gesteuerten Wechsel mit Beginn der nächsten Runden durchführen zu können. In diesen regelmäßigen Telegrammen teilt der „Master“ dem Schatten-„Master“ erstens seinen Status und zweitens seine Sicht der Dinge mit (z.B. „Membership“). Der Schatten-„Master“ dagegen antwortet mit einem „OK“, wenn er meint alles sei in Ordnung, oder mit einer Fehlernachricht. Letztere wird bei einem fehlerhaften Zustand des Schatten-„Masters“ oder eben einer abweichenden Sicht der Dinge (z.B. Rundenfehler) übertragen – dadurch werden auch viele byzantinische Fehler erkannt. Falls ein Telegramm fehlt, nimmt der beobachtende „Master“ den Ausfall des anderen „Masters“ an und teilt ihm das vorsichtshalber mit, um Buskonflikte bei sporadischen Fehlern zu unterbinden. Folglich wissen beide „Master“ stets über alles Bescheid und können angemessen reagieren. Allerdings sind die Reaktionen auf Fehler nicht immer gleich. Sie hängen davon ab, ob mehr Wert auf Sicherheit oder Verfügbarkeit gelegt wird („Fail-Safe“-Zustand, Notbetrieb, Alarm etc.), also von der Anwendung.

6.4.2 Parallelnetz

Das Parallelnetz ist die zweite Standarderweiterung im Baustein „Extended-Reliability“. Wie Bild 6.4 zeigt, repliziert diese Methode ein „Basic-TTP/A“-Netz vollständig. Dadurch existieren keine „Single-Points-of-Failure“ mehr, womit die Voraussetzung für den Einsatz in kritischen Anwendungen mit sehr hohen Sicherheits- oder Verfügbarkeitsansprüchen geschaffen ist. Aufgrund der großen Redundanz, ist die Methode Parallelnetz deutlich teurer als die Methode Schatten-„Master“. Ob sie auch teuer ist, hängt von der Realisierung ab. Wenn das Basis- und Parallelnetz diversitär und mit Spezialbauteilen aufgebaut werden, was technisch gesehen Vorteile hat, trifft das sicherlich zu. Verwendet man dagegen gleiche oder ähnliche Standardbauteile, kann diese Methode aus folgenden Gründen relativ günstig sein: Angenommen der Sensor/Aktorbus TTP/A sei weit verbreitet und werde, entsprechend seinem Konzept, häufig in „low-cost“ Systemen eingesetzt. Folglich wird es sehr viele unter-

schiedliche, einfache, günstige „Slaves“ geben. Damit verbunden wird ein großer Markt mit zahlreichen Konkurrenten und hohem Kostendruck sein. Um dem standhalten zu können, dürfen die „Slaves“ nur das Nötigste besitzen, also den Sensor oder Aktor, einen „low-cost“ µC und nur einen Busanschluss. Außerdem soll der Markt für sichere bzw. verlässliche Systeme, auch wenn er zur Zeit wächst, verhältnismäßig klein sein. Somit wird es nur wenige TTP/A-„Slaves“ mit zwei oder mehreren Busanschlüssen für den Aufbau von sicheren bzw. verlässlichen Systemen geben. Die Kosten für diese „Slaves“ werden entsprechend hoch sein. Zum einen aus Stückzahlgründen, zum anderen reichen die Ressourcen eines „low-cost“ µCs für zwei oder mehrere Busanschlüsse meistens nicht aus und schließlich, weil in einem TTP/A-„Slave“ der Sensor oder Aktor und die Buselektronik eine Einheit bilden. Aus diesen Gründen ist damit zu rechnen, dass ein Parallelnetz mit Standardbauteilen günstiger als ein redundanter Kanal mit „Spezial-Slaves“ ist.

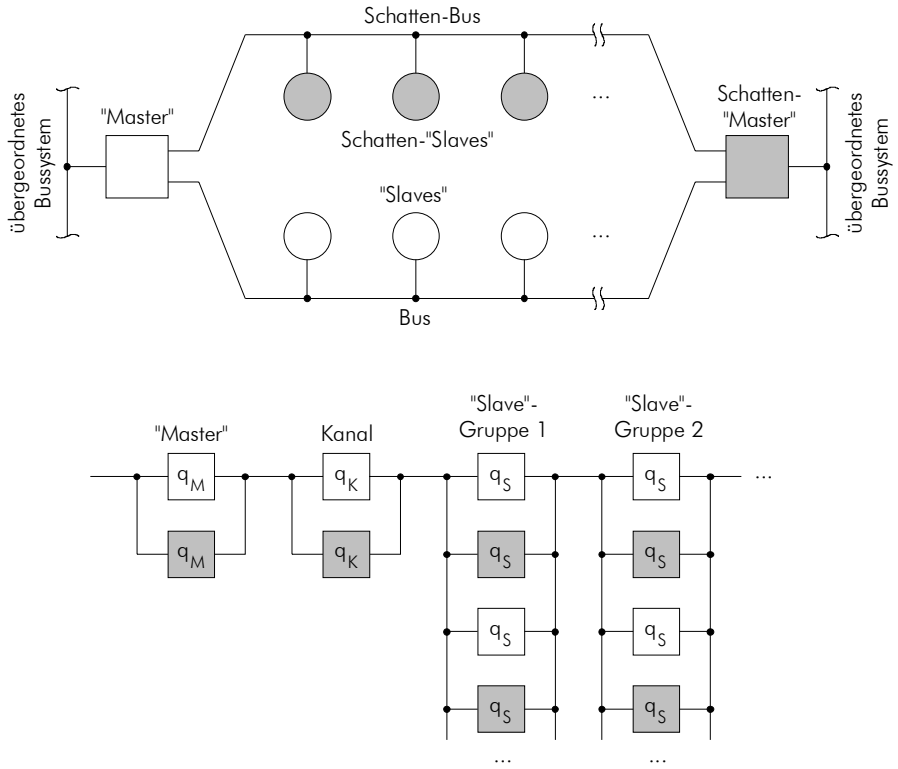


Bild 6.4: Topologie und Zuverlässigkeitsschaltbild eines „Basic-TTP/A“-Netzes mit Parallelnetz

Beim „Master“ ist die Sachlage zwar ähnlich, aber verschieden, weswegen die beiden „Master“ in Bild 6.4 einen zweiten Busanschluss besitzen. Ein „Master“ hat auf die Gesamtkosten kaum Einfluss, weil er nur ein- oder zweimal in einem TTP/A-Netz vorkommt. „Slaves“ hingegen treten n-mal auf und dominieren folglich die Gesamtkosten. Des Weiteren verfügt der „Master“ im Allgemeinen über genügend Ressourcen für einen zweiten Busanschluss. Und möglicherweise kann ein vorhandener, nicht ausgelasteter Rechner die Aufgabe des „Masters“ übernehmen.

Die Verfügbarkeit p_{PN} und die Unverfügbarkeit q_{PN} (Index PN: Parallelnetz) dieses Ansatzes sind naturgemäß besser als die Bisherigen. Verantwortlich ist nicht allein der redundante Kanal, der zwar das Gros ausmacht, sondern auch die replizierten „Slaves“. Man erkennt das besonders gut in Formel 6.6, die aus dem Zuverlässigkeitsschaltbild in Bild 6.4 abgeleitet ist. Da sämtliche Unverfügbarkeiten q_X

mindestens die Potenz zwei besitzen und wesentlich kleiner als eins sind, nimmt die Gesamtunverfügbarkeit q_{PN} drastisch ab und die Gesamtverfügbarkeit p_{PN} entsprechend zu.

$$p_{PN} = (1 - q_M^2) \cdot (1 - q_K^2) \cdot (1 - q_S^{2 \cdot g1}) \cdot (1 - q_S^{2 \cdot g2}) \cdot \ldots$$
$$q_{PN} = 1 - p_{PN}$$

$$q_M, q_K, q_S << 1, \text{ insbesondere } q_S^{2 \cdot g1}, q_S^{2 \cdot g2} \ldots$$

$$\Rightarrow \quad p_{PN} \approx (1 - q_M^2) \cdot (1 - q_K^2) \approx 1 - q_M^2 - q_K^2$$
$$q_{PN} \approx q_M^2 + q_K^2$$

Formel 6.6: Verfügbarkeit p_{PN} und Unverfügbarkeit q_{PN} mit einem Parallelnetz

Hierzu ein vergleichendes Zahlenbeispiel: Es werden die gleichen Unverfügbarkeiten $q_M = 10^{-2}$ und $q_K = 10^{-3}$ wie beim Zahlenbeispiel des Schatten-„Masters“ angenommen – aufgrund der höheren Potenz, darf man die Unverfügbarkeiten der „Slaves“ q_S hier ebenso vernachlässigen. Dementsprechend liefert Formel 6.6 für die Gesamtunverfügbarkeit das Ergebnis $q_{PN} \approx 1{,}01 \cdot 10^{-4}$. Die Verbesserung zeigt sich durch einen Vergleich mit den Gesamtunverfügbarkeiten der Schatten-„Master“-Methode $q_{SM} \approx 10^{-3}$ und des „Basic-TTP/A“-Netzes $q_{Basic} \approx 1{,}1 \cdot 10^{-2}$ (siehe Abschnitt 6.4.1). Gegenüber der Schatten-„Master“-Methode hat die Gesamtunverfügbarkeit um $(q_{PN} - q_{SM})/q_{SM} \approx -0{,}9$ abgenommen und gegenüber dem „Basic-TTP/A“-Netz um $(q_{PN} - q_{Basic})/q_{Basic} \approx -0{,}99$. Die Wirksamkeit der Parallelnetz-Methode ist also gegeben. Ihre Effizienz ist jedoch geringer als die der Schatten-„Master“-Methode. Mit den gehabten Zahlenwerten reduziert ein Schatten-„Master“ die Unverfügbarkeit eines „Basic-TTP/A“-Netzes um ca. 91 % (siehe Abschnitt 6.4.1). Ein Parallelnetz bewirkt zwar um 8 % mehr, allerdings mit wesentlich höherem Aufwand. Deshalb hat die Parallelnetz-Methode ihren eigentlichen Vorteil in der Absenz von „Single-Points-of-Failure“.

Die Rolle des Bausteins „Extended-Reliability“ besteht nun darin, das Basis- und Parallelnetz den Anforderungen und Umständen gemäß zu verwalten. Prinzipiell bestehen keine gravierenden Unterschiede zur Schatten-„Master“-Methode. Im Normalfall kontrollieren sich beide „Master“ und tauschen entsprechende Status- und Quittierungsinformationen aus. Die notwendige Kommunikation sollte, wie erörtert, am Schluss jeder Runde stattfinden. Darüber hinaus bietet das Parallelnetz neue Möglichkeiten. Je nach Anwendung, können die redundanten „Slaves“ aktiv oder passiv genutzt werden. Im Allgemeinen wird man sie aktiv redundant einsetzen, wenn es um Sicherheit geht. Hierzu müssen beide Teilnetze synchron arbeiten und einen Datenvergleich durchführen. Ersteres wird durch die Realzeitfähigkeit von „Basic-TTP/A“ unterstützt, Letzteres ist Aufgabe des Bausteins „Extended-Reliability“. Der Umgang mit den Sensor-„Slaves“ ist relativ einfach. Beide „Master“ empfangen praktisch gleichzeitig die Messwerte eines Sensorpaars. Wenn die Messwerte gleich sind oder sich bei analogen Sensoren in einem gewissen Toleranzband bewegen[129], werden sie akzeptiert und damit gerechnet. Andernfalls erfolgt eine Fehlermeldung an die Applikation, ähnlich wie bei einem Übertragungsfehler. Aufwändiger sind Aktor-„Slaves“ zu handhaben, weil sie im Fehlerfall gegensinnig agieren können. Aus diesem Grund bedarf es einerseits konstruktiver Maßnahmen, wie z.B. die Reihen- oder Parallelschaltung von Relais, einer mechanischen Kupplung, eines Interaktorabgleichs usw. An-

[129] Zusätzlich können die Daten eines analogen Sensorpaars für die Erhöhung der Messgenauigkeit genutzt werden. Zum Beispiel durch eine simple Mittelwertbildung. Dort verringert sich die Streuung typischerweise um den Faktor $1/\sqrt{2}$.

dererseits müssen die „Master“ ihre Stellwerte an jeweils beide Aktor-„Slaves“ senden, damit diese einen Vergleich durchführen können. Eine Wertetoleranz, so wie bei den Analogsensoren, ist bei den Aktoren nicht zulässig. Ferner besteht die Möglichkeit mit Rückmeldungen zu arbeiten. Das ist vom Prinzip nur eine Sache der Applikation und des „Schedules“, aufgrund der TTP/A-typischen Idempotenz aber oftmals unnötig.

Steht dagegen die Verfügbarkeit im Vordergrund, verhalten sich die redundanten „Slaves“ unter Normalbedingungen passiv. Teilweise werden sie erst im Fehlerfall eingeschaltet werden, um den betriebsbedingten Verschleiß zu minimieren oder gar zu eliminieren. Die Verwaltung der „Slaves“ ist einfacher als im vorherigen Fall. Sobald ein aktiver „Slave“ oder der Kanal ausfällt, benutzen die „Master“ eben den jeweils anderen – wegen der Ringstruktur, darf sogar ein Kanal unterbrochen und ein beliebiger „Slave“ defekt sein. In der Regel erzeugen sie dann einen Alarm und schalten auf Notbetrieb um. Mischformen, wie der Notbetrieb mit reduziertem Arbeitsbereich („Fail-Graceful“), sind natürlich auch realisierbar. Inwieweit jedoch diese Vorschläge umgesetzt werden bzw. umsetzbar sind, ist Sache des geplanten Forschungsvorhabens „Smart-Transducers“. Das Haupthindernis dürften die wenigen Ressourcen der „low-cost“ µCs sein.

6.5 Zusammenfassung

Bussysteme in verlässlichen Systemen müssen realzeitfähig sein, eine sehr gute Fehlersicherung besitzen und entweder über einen „Fail-Safe“-Zustand verfügen oder dürfen keine „Single-Points-of-Failure“ aufweisen. Damit TTP/A diese Anforderungen erfüllt, benötigt man neben dem Basisprotokoll „Basic-TTP/A“ den optionalen Baustein „Extended-Reliability“. Das Redundanzkonzept beschreibt prinzipielle Erweiterungen und Maßnahmen für diesen Baustein unter Berücksichtigung des Kostenaspekts, der ausschließlichen Verwendung von Standardbauteilen, der geringen Ressourcen von „low-cost“ µCs, einer möglichst hohen Übertragungseffizienz und der unterschiedlichen Ausprägungen von verlässlichen Systemen.

Für die Erweiterung der Fehlersicherung wurden auf „Basic-TTP/A“ bauende, ressourcen- und effizienzschonende Codierungen entwickelt. Sie verringern die Restfehlerwahrscheinlichkeit bei der Übertragung von „Fireworks-“ und Daten-„Frames“ so stark, dass die Kriterien der höchsten Datenintegritätsklasse nach DIN 19244 (I3: kritische Informationsübertragungen) auch bei relativ hohen Bitfehlerwahrscheinlichkeiten erfüllt werden – bei einer Bitfehlerwahrscheinlichkeit von 10^{-3} liegt die Restfehlerwahrscheinlichkeit unter 10^{-16}, gefordert werden nur 10^{-12}.

Zur Erhöhung der Verfügbarkeit und zur Beseitigung von „Single-Points-of-Failure“ sieht das Redundanzkonzept eine zweistufige Erweiterung der Topologie vor. In der ersten Stufe wird einem „Basic-TTP/A“-Netz ein Schatten-„Master“ hinzugefügt, weil der „Master“ in TTP/A eine zentrale Rolle spielt. Damit können verlässliche Systeme mit „Fail-Safe“-Zustand und hohem Kostendruck, wie z.B. „Airbag“-Steuerungen und Notaussysteme, realisiert werden, sofern der Übertragungskanal durch konstruktive Maßnahmen geschützt ist. Die zweite Ausbaustufe beseitigt sämtliche „Single-Points-of-Failure“ durch ein Parallelnetz. Aufgrund der großen Redundanz, eignet sich die zweite Ausbaustufe für Anwendungen mit sehr hohen Sicherheits- bzw. Verfügbarkeitsansprüchen und moderatem Kostendruck. Beispiele sind „X-by-Wire“-Subsysteme und Steuerungen in der Raumfahrt und in Zügen.

7 Resümee und Ausblick

Wichtige Ergebnisse

Sensor/Aktorbusse gewinnen durch die wachsende Komplexität von technischen Systemen und dem Wunsch nach einer durchgängigen Vernetzung an Bedeutung. Um ein breites Anwendungsspektrum abdecken zu können, muss ein moderner Sensor/Aktorbus kostengünstig und realzeitfähig sein, Sensor/Aktordaten effizient übertragen, eine gute Fehlersicherung besitzen, redundante Strukturen unterstützen, relativ hohe Baudraten zulassen und über eine standardisierte Daten-/Nachrichtenschnittstelle verfügen. Diese Eigenschaften werden selten gleichzeitig benötigt und müssen in Spezialfällen modifizierbar sein. Folglich macht es Sinn, wenn ein moderner Sensor/Aktorbus zudem skalierbar und offen ist. Existierende und neue Sensor/Aktorbusse besitzen diese Eigenschaften nur teilweise und haben Schwächen beim Realzeitverhalten, der Effizienz, Fehlersicherung, Skalierbarkeit und Offenheit, wie eine Analyse und Bewertung gezeigt hat. Das TTP/A-Konzept erfüllt hingegen alle Anforderungen.

Mit dieser Arbeit wurden Beiträge zum TTP/A-Konzept geleistet. Hierzu zählt einerseits der Grundstein für die Architektur, um TTP/A skalierbar und offen zu machen. Die Vorteile dieses für Bussysteme ungewöhnlichen Ansatzes konnten in der Zusammenarbeit mit der Universität Stuttgart und SiemensVDO bereits genutzt werden. Anderseits gehört die Prägung des Basisprotokolls „Basic-TTP/A“ zu den geleisteten Beiträgen mit dem Ziel, „Basic-TTP/A“ auf gewöhnliche Hardware-UARTs oder entsprechende Emulationen aufzusetzen, mit den knappen Ressourcen von „low-cost“ µCs auszukommen, absolut zeitdeterministisch zu sein, hocheffizient zu übertragen, eine ausreichende Fehlersicherung zu besitzen (Datenintegritätsklasse I1 nach DIN 19244) und den ereignisorientierten „Master/Slave“-Anteil angemessen zu berücksichtigen. Aus diesem Grund wurden eine implizite Synchronisation aufgenommen, eine konstante Länge für die IRG definiert, die IFG minimal kurz gehalten, eine spezielle „Fireworks“-Codierung entwickelt, sporadische Runden eingeführt, unterschiedliche Kommunikationsverbindungen festgelegt und interne Abläufe betrachtet.

Eines der wichtigsten Ergebnisse ist das Realisierungskonzept für „Basic-TTP/A“ unter der zusätzlichen Restriktion der Umsetzbarkeit in Software. Als wesentliche Probleme wurden der UART-„Jitter“, „Critical-Sections“, unterschiedlich lange Programmpfade, Latenz- und Befehlsausführungszeiten, Baudraten-abhängige Zeitfehler und daraus resultierend, die exakte Platzierung der „Fireworks-Frames“ und die Synchronität der „Slot-Timer“ identifiziert. Mit dem „Interleave“-Algorithmus, der rechenzeitorientierten „Task“-Aufteilung, der Selbstjustierung, der Lage- und, je nach Oszillatorgüte, der Gangfehlerkorrektur von „Slot-Timer“ und UART ist es gelungen alle Schwierigkeiten zu überwinden. Die Umsetzbarkeit wurde anhand der funktionierenden Implementierung für die mit Standardbauteilen aufgebauten C16Xnodes beispielhaft gezeigt. Außerdem konsolidieren die maximale Baudrate und der geringe Ressourcenbedarf das Realisierungskonzept. Der 10 MIPS starke C16x-µC kann das „Basic-TTP/A“-Protokoll nebst Anwendung bis 625 kBit/s (Grenze des C16x-µCs) ausführen. Er benötigt dafür eine unterbrechbare CPU, 10..1000 Bytes flüchtigen Speicher (anhängig vom Knotentyp und der RODL-Größe), 1..1,5 KB nicht flüchtigen Speicher (anhängig vom Typ und Aus-

bau eines Knotens), einen „Timer“ mit einer Minimalauflösung von $T_{Bit, M}/8$, ein Hardware-UART mit mindestens achtfacher Bitteilung oder eine entsprechende UART-Emulation und 2..4 I/O-Pins (abhängig vom UART-Typ und „Physical-Layer“). Da die meisten „low-cost“ µCs o.Ä. und UARTs diese Voraussetzungen erfüllen, ist das „Basic-TTP/A“-Konzept allgemeingültig.

Weiterhin wurde ein Redundanzkonzept für den optionalen TTP/A-Baustein „Extended-Reliability“ erarbeitet. „Basic-TTP/A“ ist für einfache und realzeitkritische, aber nicht für verlässliche Anwendungen geeignet. Nichtsdestotrotz kann man es mit entsprechenden Maßnahmen dahingehend erweitern – bei anderen Bussystemen ist das in der Regel nicht möglich. Das Redundanzkonzept beschreibt diese Maßnahmen prinzipiell, zeigt deren Wirksamkeit mittels Berechnungen und berücksichtigt die geringen Ressourcen von „low-cost“ µCs, vermeidet zusätzliche Bauteile, schont die Übertragungseffizienz und beachtet die unterschiedlichen Ausprägungen von verlässlichen Systemen. Zum einen wird mit speziellen, effektiven Codes die Fehlersicherung der „Fireworks-“ und Daten-„Frames“ so stark erhöht, dass die Kriterien der höchsten Datenintegritätsklasse nach DIN 19244 (I3: kritische Informationsübertragungen) auch bei relativ hohen Bitfehlerwahrscheinlichkeiten ($> 10^{-3}$) erfüllt werden. Zum anderen steigert eine zweistufige Erweiterung der Topologie die Verfügbarkeit von „Basic-TTP/A“. Die erste Stufe sieht einen Schatten-„Master“ vor und eignet sich für verlässliche Systeme mit „Fail-Safe“-Zustand und hohem Kostendruck (z.B. „Airbag“-Steuerungen, Notaussysteme ...), sofern der Übertragungskanal angemessen geschützt ist. In der zweiten Stufe beseitigt ein Parallelnetz sämtliche „Single-Points-of-Failure“. Aufgrund dieser Eigenschaft und der großen Redundanz, eignet sich die zweite Stufe für Anwendungen mit sehr hohen Sicherheits- bzw. Verfügbarkeitsansprüchen und moderatem Kostendruck (z.B. „X-by-Wire“-Subsysteme, Steuerungen in der Raumfahrt und in Zügen ...).

Zukünftige Forschungsaktivitäten

Aufgrund der Erfolge in dem Forschungsprojekt TTSB, des Interesses seitens der Industrie, der OMG-Standardisierung von TTP/A und der effizienten Kooperation zwischen den Projektpartnern, ist ein weiteres Forschungsvorhaben unter dem Namen „Smart-Transducers“ geplant [19]. In diesem Forschungsvorhaben soll zum einen der Baustein „Basic-TTP/A“ ausgebaut werden. Wie beschrieben, existiert für ihn eine universelle Referenzimplementierung und eine Portierung für C16x-µCs in der Hochsprache C. Von der universellen Referenzimplementierung sollen spezialisierte Implementierungen abgeleitet werden, um den Speicherbedarf zu minimieren. Das ist insbesondere für „Slaves“ interessant, weil sie i.A. über „low-cost“ µCs mit entsprechend wenig Ressourcen verfügen. Ein einfacher binärer Sensor-„Slave“ z.B. wird selten mehr als zwei bis drei „Slots“ belegen, so dass mit einer Spezialisierung der Speicherbedarf von derzeit ca. 1,5 KB auf 1 KB oder weniger reduzierbar sein dürfte. Damit die Vielfalt der Spezialisierungen überschaubar bleibt, empfiehlt sich eine Klassifizierung z.B. nach der Anzahl benötigter „Slots“. Außerdem ist die Portierung von „Basic-TTP/A“ auf „low-cost“ µCs, wie z.B. den PICs von Microchip oder den AVRs von Atmel, vorgesehen. Hierbei wird die existierende C16x-Portierung als Vorlage wertvolle Dienste leisten.

Ein weiterer Punkt ist der Einsatz eines Realzeitbetriebssystems. Er ist hauptsächlich für den „Master“ interessant, da dieser Knoten in der Regel die meistens Aufgaben erledigt und über die erforderlichen Ressourcen verfügt. Mit einem entsprechenden Realzeitbetriebssystem auf dem „Master“ kann man sich z.B. die Programmierung eines TCP/IP-Stacks für ein „Ethernet-Gateway“ oder eines „Web-Servers“ für eine moderne Prozesskontrolle sparen. Zuvor muss jedoch ein generelles Problem gelöst werden. Der gleichzeitige Betrieb von „Basic-TTP/A“ und einem Realzeitbetriebssystem auf einer gemeinsamen CPU schafft einen Ressourcen- und gegebenenfalls einen Realzeitkonflikt. Weil in diesem Fall „Basic-TTP/A“ die eindeutig höhere Priorität hat, muss ein Realzeitbetriebssystem in „Basic-TTP/A“ quasi eingebettet werden. Vor allem aber gilt es eine Lösung für die typischen „Critical-Sections“ im „Kernel“ eines Realzeitbetriebssystems zu finden. Als Vorbereitung wurden ein für For-

schungszwecke kostenloses Realzeitbetriebssystem im Quellcode beschafft (µCOS II), eine weitestgehend in C gehaltene Portierung für die C16x-µCs geschrieben und die „Mulitasking“-Unterstützung in „Basic-TTP/A“ integriert.

Zum anderen sollen in dem Forschungsvorhaben „Smart-Transducers“ die Bausteine „Interface-File-System“ und „Extended-Reliability“ weiterentwickelt bzw. geschaffen werden. Für das teilweise existierende „Interface-File-System“ sind Erweiterungen hinsichtlich „Plug&Play“ und „Online“-Konfiguration geplant. Die notwendigen Voraussetzungen, insbesondere die „Hot-Plug-In“-Fähigkeit, hält die „Basic-TTP/A“-Implementierung bereit. In diesem Zusammenhang ist auch die Fortführung der Arbeiten an der Internet-Anbindung vorgesehen. Es steht z.B. die Erprobung des RODL-„Tools“ der Uni Stuttgart an, mit dem man ein TTP/A-Netz komfortabel von einem Internet-„Browser“ aus konfigurieren können wird. Ferner soll das Redundanzkonzept dieser Arbeit in die Realisierung des Bausteins „Extended-Reliability“ einfließen. Um seine Wirksamkeit nachzuweisen, sind Untersuchungen im EMV-Labor und Eignungstests für Kraftfahrzeuge mit dem Standard- und „High-Speed-Physical-Layer“, unter der Federführung von SiemensVDO, angedacht.

„Carrier-Sense“-TDMA

Abschließend möchte der Autor eine Idee vorstellen, die es möglicherweise Wert ist im Forschungsvorhaben „Smart-Transducers“ weiterzuverfolgen. Es handelt sich um eine noch ressourcenschonendere Variante der vorgestellten TDMA-„Master/Slave“-Arbitrierung. Das neue Verfahren trägt den Namen „Carrier-Sense“-TDMA (kurz CS-TDMA) und hat seine Funktionstüchtigkeit in einer rudimentären Form bereits bewiesen. Auslöser für die folgenden Überlegungen waren die erheblichen Schwierigkeiten bei der Erarbeitung einer Lösung zur Einhaltung der Realzeitbedingungen. „Basic-TTP/A“ arbeitet zeitgenau und besitzt eine hohe Wiederholgenauigkeit. Davon profitieren insbesondere „Slaves“ mit ungenauen RC- oder „On-Chip“-Oszillatoren. Sie arbeiten mit fast der gleichen Zeitpräzision wie der quarzgesteuerte „Master“. Dementsprechend komplex und umfangreich sind die Synchronisationsalgorithmen ausgefallen. Wenn man auf die Wiederholgenauigkeit, also die exakte Platzierung der „Frames“ in den „Slots“, verzichten kann, lassen sich die Synchronisationsalgorithmen deutlich vereinfachen. Dadurch wird in erster Linie der Speicherbedarf reduziert, was einer Implementierung auf „low-cost“ µCs der untersten Kategorie, wie z.B. der PIC12Cx-Familie, entgegenkommt.

Beim CS-TDMA-Verfahren tritt anstelle der Zeitsteuerung des „normalen“ TDMA-Verfahrens eine Folgesteuerung. Aufgrund dessen verhält sich Ersteres kooperativ, wohingegen Letzteres restriktiv handelt. Ein CS-TDMA-Teilnehmer belegt keinen Zeitschlitz, sondern er wartet bis sein Vorgänger fertig ist. Dazu hört er den Bus ständig ab („Carrier-Sense“) und registriert mit einem Zähler die empfangenen „Frames“ – beim „normalen“ TDMA-Verfahren ist das nicht erforderlich. Sobald der Zählerstand einen gewissen Wert erreicht, reagiert der CS-TDMA-Teilnehmer, indem er das letzte „Frame“ verarbeitet oder zu senden beginnt. Die hierfür notwendigen Informationen sind, wie beim „normalen“ TDMA-Verfahren, in einem „Schedule“ gespeichert. Wegen dieser Arbeitsweise, benötigt man beim CS-TDMA-Verfahren kein „Slot-Timer“. Folglich ist die aufwändige Synchronisation einer Zeitsteuerung mit asynchroner Übertragung nicht erforderlich, so dass die beschriebene Folgesteuerung wesentlich einfacher umsetzbar ist. Neben dem geringen Ressourcenbedarf, hat das CS-TDMA-Verfahren einen weiteren Vorteil. Durch sein kooperatives Verhalten nimmt es sowohl auf Verspätungen als auch auf Verfrühungen seiner Vorgänger Rücksicht. Aus diesem Grund können beim CS-TDMA-Verfahren erstens keine Buskonflikte entstehen. Und zweitens werden unnötige „Interframe-Gaps“ vermieden, wodurch die Übertragungseffizienz um bis zu ca. 5 % steigen kann.

Der bislang einzige Nachteil dieses neuen Verfahrens ist die Fortpflanzung von Zeitfehlern. Aufgrund des kooperativen Verhaltens, akkumulieren sich die Längenfehler der UART-„Frames“. Je weiter weg sich ein „Frame“ vom Beginn einer Runde befindet, desto ungenauer ist seine Platzierung. Da die UART-„Frames“ nicht beliebig lang oder kurz sein dürfen, lässt sich der relative Platzierungsfehler quantisieren. Für die UART-„Frames“ gilt weiterhin das Toleranzband ±1,84 %, bezogen auf den „Master“-Takt. Unter Hinzunahme von Reaktions- und Verarbeitungszeiten in den Knoten, kann der relative Platzierungsfehler schätzungsweise bis auf ±2 % anwachsen. Das ist zwar deutlich schlechter als beim jetzigen „Basic-TTP/A“, aber immer noch viel besser als die Frequenztoleranz eines RC- oder „On-Chip“-Oszillators. Um das Anwachsen des absoluten Zeitfehlers zu begrenzen, wird das CS-TDMA-Verfahren nur in den Runden angewandt. Außerhalb der Runden gelten die alten Regeln. Da der „Master“ die „Fireworks-Frames“ also nach wie vor mit der gleichen Präzision und im gleichen Zeitraster sendet, ist die Realzeitfähigkeit des Bussystems mit dem CS-TDMA-Verfahren ebenfalls gewährleistet. Wie beim „normalen“ TDMA-Verfahren, kann man genau berechnen, in welchem Zeitraum ein „Frame“ erscheint. Nur, dass beim CS-TDMA-Verfahren diese Zeiträume großzügiger ausfallen. Dadurch sinkt die Wiederholgenauigkeit bzw. steigt der „Frame-Jitter“. In den sehr vielen Anwendungen, wo das irrelevant ist, dürfen die beiden Arbitrierungsmethoden bezüglich des Zeitverhaltens jedoch als gleichwertig angesehen werden.

Natürlich gibt es auch ein zu lösendes Problem mit dem CS-TDMA-Verfahren. Wenn ein „Slave“ ausfällt, gerät der Mechanismus ins Stocken. Hier können z.B. Pseudo-„Frames“ vom „Master“ oder „Timeouts“ in allen Knoten Abhilfe schaffen. Bei der ersten Methode muss der „Master“ die Startbitflanken am Bus überwachen und nach einer Zeitüberschreitung ein sicher erkennbares Pseudo-„Frame“ senden. Für den „Master“ ist das kein Problem, weil er im Allgemeinen genügend Ressourcen besitzt („Interrupt-Pin“, „Timer“ und Rechenzeit). Dagegen ersetzt die zweite Methode den fehlenden „Receive-Interrupt“ durch einen „Timer-Interrupt“. Es ist anzunehmen, dass die Pseudo-„Frame“-Methode kleinere Zeitfehler produziert, weil der „Master“ die Zeitreferenz ist. Ob das tatsächlich zutrifft, und ob diese Methode auch insgesamt besser ist, muss eine Analyse zeigen.

8 Anhang

8.1 Zeit- contra Ereignissteuerung bei Bussen

Aufgrund des gewachsenen Sicherheitsbedürfnisses bzw. -bewusstseins, hat der Realzeitaspekt bei den Bussystemen zugenommen. Das führt leider immer wieder zu leidenschaftlichen Diskussionen um die Art der Realzeitsteuerung. Aus diesem Grund werden hier kurz die Vor- und Nachteile der beiden Kontrahenten, der Zeit- und Ereignissteuerung, für Bussysteme erörtert. Fakt ist, dass bei den Bussystemen ein Trend zur Zeitsteuerung für sicherheitsrelevante Anwendungen besteht (TTP/C, FlexRay, ProfiSafe, SafetyBus...). Das heißt jedoch nicht, dass eine Ereignissteuerung dafür ungeeignet ist. Ansonsten müsste man auch die Realzeitfähigkeit von Realzeitbetriebssystemen in Frage stellen. Dort dominieren nämlich die Ereignissteuerungen, und zwar mit nachgewiesener Realzeitfähigkeit (RMP, EDF...) [136]. Folglich eignen sich prinzipiell beide Verfahren für Realzeitanwendungen.

Zeitsteuerungen arbeiten nach einem festen Schema, dem so genannten „Schedule“. Dieser „Schedule“ wird vorab festgelegt und berücksichtigt alle Eventualitäten. Seine Existenz ist der Nachweis für die Realzeitfähigkeit eines Systems. Allerdings kann die Erstellung eines „Schedules“ sehr komplex sein, da es sich um ein NP-hartes Problem handelt [136]. Deshalb verwendet man für das „Scheduling“ oft rechnergestützte Heuristiken. Die A-Priori-Definition des Laufzeitverhaltens begünstigt eine ressourcenarme Umsetzung zeitgesteuerter Systeme (Listensteuerung). Für einen Bus bedeutet das, dass er weder eine aufwändige Arbitrierung noch eine Kollisionsauflösungsstrategie benötigt. Als Folge haben zeitgesteuerte Busse i.A. einfachere Protokolle wie ereignisgesteuerte Busse, weshalb sie sich für eine Implementierung auf „low-cost“ µCs besser eignen. Allerdings erfahren ereignisverknüpfte Telegramme, wegen des statischen Verhaltens zeitgesteuerter Busse, eine mittlere Verzugszeit in Höhe der halben Sendeperiode. Ferner bewirken Telegramme, die von nichtperiodischen Ereignissen abhängen eine ungünstige Busauslastung. Denn die Telegrammperiode ist stets so zu wählen, dass sie kleiner als der kürzeste Ereignisabstand ist. Bei „Burst“-artigen Ereignissen werden dadurch viele unnötige Telegramme übertragen. Auf der anderen Seite verhindern diese idempotenten Telegramme permanente Übertragungsfehler. Dadurch erreichen zeitgesteuerte Bussysteme in der Regel mindestens die Datenintegritätsklasse I1 nach DIN 19244.

Bei Ereignissteuerungen ist das Verhältnis zwischen A-Priori- und Laufzeitrechenarbeit umgekehrt. Die Entscheidung welche „Task“ als Nächste rechnen darf, bzw. welcher Teilnehmer als nächster senden darf, wird im Betrieb getroffen. Teilweise sind die Entscheidungsregeln sehr aufwändig, weshalb eine Ereignissteuerung i.A. mehr Ressourcen als eine Zeitsteuerung benötigt. Aus diesem Grund ist die Implementierung eines ereignisgesteuerten, realzeitfähigen Bussystems auf „low-cost“ µCs nur schwer möglich. Im Voraus müssen dagegen nur Prioritäten, „Deadlines“ oder Rechenzeiten angegeben werden. Deshalb ist die Komplexität zwischen den beiden Methoden nicht unterschiedlich, sondern nur anders aufgeteilt. Vorteile ereignisgesteuerter Busse sind ihre Dynamik und ihre oftmals

leichtere Konfigurierbarkeit. Dadurch ist die Busbelastung nie größer als nötig, sind die Telegrammlatenzzeiten minimal kurz und die Inbetriebnahme und Wartung in der Regel komfortabler. Gegen Übertragungsfehler müssen sich ereignisgesteuerte Busse separat und besser absichern, falls sie eine Datenintegritätsklasse erreichen sollen. Abschließend sei erwähnt, dass weder zeit- noch ereignisgesteuerte Realzeitbusse bei korrekter Auslegung überlastbar sind – andernfalls sind es keine Realzeitbusse.

	Zeitsteuerung	**Ereignissteuerung**
Realzeitfähigkeit	ja	ja
Buszuteilung,	statisch	dynamisch
Busauslastung	nur bei periodischen Ereignissen gut	immer gut; passt sich dem Bedarf an
Realzeitnachweis	einfach; implizit mit dem „Schedule“	aufwändig; explizite Methoden
„Scheduling“	aufwändig; NP-hart; Heuristiken	einfach; z.B. Prioritätsvergabe
Laufzeitsystem	einfache Listensteuerung	komplexes Betriebssystem
Flexibilität	niedrig; stets neue „Schedules“ nötig	hoch, weil ereignisorientiert
Übertragungsfehler	Idempotenz + Telegrammsicherung	Telegrammsicherung

Tabelle 8.1: Vor- und Nachteile von zeit- und ereignisgesteuerten Bussen

8.2 Kanalmodell für Restfehlerabschätzungen

Für die Berechnung der Restfehlerwahrscheinlichkeit einer Kommunikation benötigt man ein mathematisches Modell des Übertragungskanals. Je besser dieses Kanalmodell mit der Realität übereinstimmt, desto vertrauenswürdiger ist die Restfehlerwahrscheinlichkeit. Es gibt allerdings kein Kanalmodell, dass für alle „Physical-Layer“ und Umgebungsbedingungen zutrifft. Ein Kupferkabel hat eine andere Charakteristik als eine Glasfaser oder eine Funkstrecke. Des Weiteren ist eine unsymmetrische Übertragung mit kleinen Signalpegeln empfindlicher als eine differentielle Übertragung mit großen Signalpegeln. Und schließlich sind die Störquellen in einem Kfz, einer Industriemaschine, einer Werkstatt, einem Haus oder einem Büro zueinander und auch von Fall zu Fall unterschiedlich. Aus diesen Gründen können die gesuchten Bitfehlerverteilungen, die oftmals messtechnisch ermittelt werden, stark variieren. „Basic-TTP/A“ soll jedoch ein universeller Sensor/Aktorbus sein. Folglich kann seine allgemeine Restfehlerwahrscheinlichkeit nur abgeschätzt werden.

Trotzdem braucht man eine, wenn auch nur approximative, mathematische Beschreibung des Übertragungskanals. Das Gilbert-Elliot-Modell (kurz GE-Modell) ist für diesen Zweck prinzipiell geeignet [104]. Es berücksichtigt „Burst“-Fehler, für die eine drahtgebundene Kommunikation in der Nähe entsprechender Störquellen (z.B. Bürstenfeuer von Motoren, Prellen von Relaiskontakten, PWM-Leistungsverstärker...) empfänglich ist. Bestandteile des GE-Modells sind eine Markovkette erster Ordnung und ein BSC-Modell („Binary-Symmetric-Channel“) mit variabler Bitfehlerwahrscheinlichkeit. Gemäß Bild 8.1, hat die Markovkette die Zustände „bad“ und „good“. Im Zustand „bad“ ist die Bitfehlerwahrscheinlichkeit des BSC-Modells $W = W_{bad}$, wohingegen für den Zustand „good“ $W = W_{good}$ gilt. Wenn $W_{bad} >> W_{good}$ ist ($W_{bad} \approx 10^{-3}$, $W_{good} \approx 10^{-6}..10^{-5}$), treten im Zustand „bad“ deutlich mehr Bitfehler auf als im Zustand „good“. Die Markovkette bewirkt, dass die Zustände zufällig wechseln, wodurch die „Burst“-Fehler entstehen. Über die mittlere Verweildauer in den Zuständen,

bzw. über die Zustandswahrscheinlichkeiten P_{good} und P_{bad}, entscheiden die Übergangswahrscheinlichkeiten P_{gb} und P_{bg}.

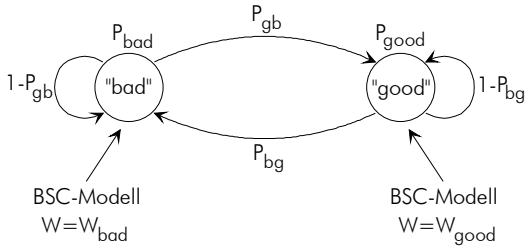


Bild 8.1: Zustandsübergangsgraph des Gilbert-Elliot-Modells

Unter der Annahme einer ergodischen Markovkette, schließlich ist soll die Restfehlerwahrscheinlichkeit nur allgemein abgeschätzt werden, sind die Zustandswahrscheinlichkeiten P_{good} und P_{bad} konstant. Vor allem aber können sie aus den Übergangswahrscheinlichkeiten P_{gb} und P_{bg} berechnet werden [102]. Für den dargestellten Fall gilt:

$$\vec{P} = \left(\underline{Q}^T + \underline{U} - \underline{E}\right)^{-1} \cdot \vec{I}$$

$$\text{mit} \qquad \vec{P} = \begin{pmatrix} P_{bad} \\ P_{good} \end{pmatrix} \qquad \vec{I} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$\text{und} \qquad \underline{Q}^T = \begin{bmatrix} 1-P_{gb} & P_{bg} \\ P_{gb} & 1-P_{bg} \end{bmatrix} \qquad \underline{U} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \qquad \underline{E} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\Rightarrow \qquad \vec{P} = \frac{1}{P_{gb} + P_{bg}} \cdot \begin{pmatrix} P_{bg} \\ P_{gb} \end{pmatrix} = \begin{pmatrix} P_{bad} \\ P_{good} \end{pmatrix}$$

Formel 8.1: Zustandswahrscheinlichkeiten im GE-Modell mit ergodischer Markovkette

Damit lässt sich die Restfehlerwahrscheinlichkeit für ein n-Bit langes „Frame“ wie folgt berechnen:

$$P_{Restfehler}(n) = \sum_e \left(P_{good} \cdot \binom{n}{e} \cdot W_{good}^e \cdot (1 - W_{good})^{n-e} + P_{bad} \cdot \binom{n}{e} \cdot W_{bad}^e \cdot (1 - W_{bad})^{n-e} \right) \cdot P_{Red}(n, e)$$

Formel 8.2: Restfehlerwahrscheinlichkeit mit einem ergodischen GE-Modell

e ist die Anzahl der relevanten Bitfehler und $P_{Red}(n,e)$ die Reduktionsfunktion, mit der berücksichtigt wird, dass nicht alle Kombinationen bei einer bestimmten Bitfehleranzahl unerkannt bleiben – bei vielen Codes ist $P_{Red}(n,e) \approx 2^{-k}$ (k: Anzahl der Kontrollstellen) [108]. Wenn nun $e >> 1$ (z.B. CRC-Codierung) und $W_{bad} >> W_{good}$ (Kanal mit „Burst“-Fehlern) ist, dann liefert der „Good“-Anteil kaum einen Beitrag zur Restfehlerwahrscheinlichkeit. Selbst für den Fall $P_{good} >> P_{bad}$ (d.h. $P_{gb} >> P_{bg}$) nicht,

weil der exponentielle Term W^{e}_{good} deutlich stärker gewichtet. Also darf man Formel 8.2, unter den geschilderten Voraussetzungen, durch Vernachlässigung des „Good"-Anteils vereinfachen:

$$P_{Restfehler}(n) \approx \frac{P_{bg}}{P_{gb}} \cdot \sum_{e} \binom{n}{e} \cdot W_{bad}^{e} \cdot (1 - W_{bad})^{n-e} \cdot P_{Red}(n, e) \qquad \text{für} \begin{cases} e >> 1 \text{ und} \\ W_{bad} >> W_{good} \text{ und} \\ P_{gb} >> P_{bg} \end{cases}$$

Formel 8.3: Vereinfachte Restfehlerwahrscheinlichkeit mit einem ergodischen GE-Modell

Formel 8.3 ist prinzipiell jedoch nichts anderes als die Restfehlerwahrscheinlichkeit eines BSC-Modells. Damit ist gezeigt, dass für Abschätzungen der Restfehlerwahrscheinlichkeit das einfachere BSC-Modell mit einer entsprechend hohen Bitfehlerwahrscheinlichkeit genügt. Sofern der Quotient $P_{bg}/P_{gb} << 1$ vernachlässigt wird, handelt es sich um eine „Worst-Case"-Abschätzung.

8.3 UART-Emulationen für TTP/A

Obwohl der Sensor/Aktorbus TTP/A auf Hardware-UARTs basiert, arbeitet er natürlich auch mit UART-Emulationen. Davon sind die so genannten Software-UARTs besonders interessant. Software-UARTs bilden mit einem Software-Treiber[130], einem digitalen Ausgang, einem digitalen Eingang und gegebenenfalls einem „Interrupt"-Eingang das Verhalten eines Hardware-UARTs nach. Sie erschließen für TTP/A, vor allem für „Basic-TTP/A", die unterste Preiskategorie der „low-cost" µCs. Potenzielle Beispiele sind die Bausteine PIC12Cx und PIC16Cx von Microchip oder die Bausteine ATtiny-x und AT90Sx von Atmel. Diese smarten µCs, mit zum Teil nur acht Pins, haben einen internen Oszillator, einen „Timer", eine „Watchdog", ein paar digitale I/Os und je nach Typ einen A/D-Wandler. Außerdem weisen viele einen breiten Betriebsspannungsbereich zwischen ca. 2,5..5,5V auf. Für einen TTP/A-Betrieb fehlt ihnen lediglich ein Software-UART.

	Hardware-UART	**Software-UART**
Baudrate	bis einige MBit/s	max. 10..30 kBit/s
Granularität der Baudrate	fein bei niedrigen Baudraten, grob bei hohen Baudraten	sehr fein bis fein im gesamten Baudratenbereich
Ressourcen	interner oder externer UART	ein digitaler Ausgang, ein digitaler Eingang, ggf. ein „Interrupt-Pin"; Speicher
CPU-Last	sehr gering (‰)	sehr hoch (30% und mehr)
„Frame-Jitter"	groß (Ausnahme „resetable" UARTs)	gering
„Fireworks-Jitter"	mit entsprechender Lösung Null	wie „Frame-Jitter"
Mehrfachabtastung	sehr oft 3-fach Abtastung zur Erhöhung der Störsicherheit	wegen zusätzlicher Rechenlast und Komplexität normalerweise nicht
„Slave"	günstig, schnell	sehr günstig, langsam
„Master"	günstig, schnell	in Ausnahmefällen (z.B. TPU oder SPI)

Tabelle 8.2: Vor- und Nachteile von Hard- und Software-UARTs für TTP/A

[130] Hersteller und „User-Groups" stellen im Internet für viele µCs entsprechende Treiber zum „downloaden" bereit.

Dieser bringt jedoch auch Nachteile mit sich. Gegenüber einem Hardware-UART benötigt er zusätzlich Codespeicher, etwas Datenspeicher und verursacht eine nicht zu unterschätzende Menge an Rechenarbeit. Da ein Software-UART nicht Zweck, sondern Mittel ist, beschränkt Letzteres die max. Baudrate auf den unteren 10 kBit/s-Bereich. Schließlich liegen die Rechenleistungen dieser µCs mit höchstens 3..5 MIPS, so wie ihre Kosten, in der untersten Klasse. Weitere Vor- und Nachteile sind der Tabelle 8.2 zu entnehmen. In der Regel ist ein Software-UART nur bei sehr kostensensitiven, einfachen „Slaves“ (Schalter, Relais, Temperaturfühler...) sinnvoll. Gerade der Kostenaspekt macht diese „Slaves“ aber in vielen Anwendungen mit geringen Ansprüchen attraktiv. Ein paar Beispiele sind Tür- und Sitzsteuerungen in Kfzs, Bedien- und Beobachtungselemente an Automaten und Lichtsteuerungen und Temperaturregelungen in Gebäuden.

Neben einem Software-UART existieren selbstverständlich weitere Lösungen zur Emulation von einem Hardware-UART. Allerdings eignen sich diese kaum für einen „Slave“, weil sie Hardware voraussetzen, über die kein „low-cost“ µC verfügt. In Sonderfällen können sie jedoch z.B. für einen „Master“ interessant sein, weshalb im Folgenden zwei Möglichkeiten, ohne nähere Erklärung, aufgezählt werden. Etwas größere µCs besitzen des Öfteren eine SPI („Serial-Peripheral-Interface“) und/oder TPU („Time-Processing-Unit“). Mit einer entsprechenden Verschaltung und etwas Software gestattet jede dieser Komponenten eine effiziente und schnelle UART-Emulation. Details hierüber findet man z.B. auf den „Web“-Seiten von Infineon und Motorola.

8.4 Oszillatorenvergleich

Quarzoszillatoren

Typische Standardquarzoszillatoren besitzen einen Quarz mit AT-Kristallschnitt (AT-Quarz) als Resonator. Sie weisen einen relativen Abgleichfehler innerhalb eines Intervalls von ca. ±50 ppm auf. Ihr relativer Temperaturfehler hat ein symmetrisches Profil und bewegt sich in einem Betriebstemperaturbereich von -20..70°C zwischen etwa ±30 ppm. Der relative Alterungsfehler von AT-Quarzen beträgt ungefähr ±5 ppm pro Jahr. Genauere, jedoch auch teurere Quarzoszillatoren verwenden Quarzresonatoren mit einem BT-, CT- oder DT-Kristallschnitt. Die Grenzen ihrer relativen Abgleichfehler liegen zum Teil nur bei ±5 ppm. Allerdings sind ihre Temperaturprofile unsymmetrisch (negativ parabolisch) und ihre Alterungsfehler größer als bei einem AT-Quarz [129] [135]. Wegen des sehr geringen Durchgriffs der Betriebsspannung auf die Frequenz, wird die Kurzzeitfrequenzdrift von Quarzoszillatoren hauptsächlich durch Temperaturschwankungen bestimmt. Die Zeitkonstante liegt, je nach den Massen und Wärmewiderständen, im s- bis 10 s-Bereich.

Keramikoszillatoren

Falls die Genauigkeitsanforderungen weniger hoch sind, stellen Oszillatoren mit Keramikresonatoren eine preiswerte Alternative zu Quarzoszillatoren dar. Gewöhnliche Keramikresonatoren haben einen typischen relativen Frequenzabgleichfehler im Intervall ±0,5 % [131]. Damit sind sie um den Faktor 100 schlechter als Standardquarzoszillatoren. Gegenüber RLC-Oszillatoren sind sie jedoch um den Faktor 10 besser und liegen kostenmäßig nicht viel darüber. Mit der Temperaturempfindlichkeit sieht es ähnlich aus. Im Temperaturbereich von -20..80°C weisen Keramikresonatoren einen relativen Fehler von ca. ±0,3 % auf [130], wodurch sich ungefähr dieselben Zahlenverhältnisse zu Quarz- und RLC-Oszillatoren einstellen wie beim Abgleichfehler. Aufgrund der mechanischen Resonanz, ist der Betriebsspannungsdurchgriff bei Keramikoszillatoren vernachlässigbar. In Sachen Gewicht und Baugröße sind Keramikresonatoren um etwa den Faktor 0,5 leichter bzw. kleiner als Quarzresonatoren. Für RLC-Resonatoren gilt ein Faktor von etwa 0,1. Die Zeitkonstante für temperaturbedingte Fre-

quenzänderungen dürfte ungefähr so groß wie die der Quarzoszillatoren sein. Schließlich ist sowohl die Masse als auch die Baugröße – und damit die Oberfläche – von Keramikresonatoren kleiner. Von den Herstellern werden Keramikresonatoren insbesondere für „low-cost“ µC-Systeme beworben.

RC-Oszillatoren

RC-Oszillatoren ermöglichen eine einfache und günstige Takterzeugung. Aufgrund der großen Bauteiletoleranzen, vor allem bei den Kondensatoren, sind RC-Oszillatoren jedoch ungenau. Ihr Abgleichfehler kann zwar händisch oder automatisch per Lasertrimmung nachjustiert werden, allerdings ist der Kostenvorteil gegenüber Keramikozillatoren dann fraglich. Deshalb werden hier nur unkalibrierte RC-Oszillatoren untersucht. Die Frequenz eines RC-Oszillators mit nur einem frequenzbestimmenden RC-Glied gehorcht im Allgemeinen der Gleichung $f = k/RC$. k ist eine schaltungstechnische Konstante und RC die maßgebliche Zeitkonstante. Über die partiellen Ableitungen $\partial f/\partial R = -k/R^2C = -f/R$ und $\partial f/\partial C = -k/RC^2 = -f/C$ gelangt man zu den Empfindlichkeiten von f gegenüber R und C. Daraus wird das totale Differential gebildet, anschließend etwas umgeformt und die Differentiale durch Differenzen ersetzt. Als Resultat erhält man die Approximationsformel $\Delta f/f \approx -(\Delta R/R + \Delta C/C)$, mit der im Folgenden der relative Abgleich- und Temperaturfehler von einfachen RC-Oszillatoren ermittelt wird.

Gute Widerstände mit einer Toleranz von ±1 % und gute Kondensatoren mit einer Toleranz von ±5 % sind mittlerweile so günstig geworden, dass man sie in „low-cost“ Oszillatoren verbauen kann. Mit diesen Voraussetzungen beläuft sich der relative Abgleichfehler von einfachen RC-Oszillatoren, gemäß der obigen Approximationsformel, auf ca. ±6 %. Die Abschätzung der temperaturbedingten Frequenzdrift ist etwas aufwändiger, weil die relativen Temperaturfehler von Widerständen und Kondensatoren erst berechnet werden müssen. In guter Näherung gilt für das Temperaturverhalten vieler Widerstände und Kondensatoren $X_\upsilon = X_{20}\cdot(1+\alpha_X\cdot\Delta\upsilon)$. Hierbei steht X für R oder C, $\Delta\upsilon$ für den Temperaturunterschied bezogen auf 20°C und α_X für den jeweiligen Temperturbeiwert. Aus der Gleichung lässt sich der relative Temperaturfehler $\Delta X/X_{20} = (X_\upsilon - X_{20})/X_{20} = \alpha_X\cdot\Delta\upsilon$ ableiten. Das Problem einer allgemeinen Quantifizierung besteht nun darin, dass α_X stark von den Materialeigenschaften des Bauelements abhängt [135].

Keramikkondensatoren haben des Öfteren, aber eben nicht immer, einen geringeren Temperaturgang als Kunststoffkondensatoren. Die α_C-Werte unterscheiden sich betragsmäßig zum Teil um den Faktor 10..100. Ferner ist α_C einmal positiv und ein anderes mal negativ. Bei den Widerständen ist die Sachlage ähnlich, aber weniger dramatisch. Metallwiderstände besitzen meistens PTC-Eigenschaften ($\alpha_R > 0$) und Kohlewiderstände meistens NTC-Eigenschaften ($\alpha_R > 0$). Darüber hinaus ist das Verhältnis der α_R-Werte in der Regel nicht größer als 10. Wenn man sich jedoch auf die besseren Metallfilmwiderstände und NDK-Keramikkondensatoren (niedrige Dielektrizitätskonstante) bzw. Polystyrol-Folienkondensatoren beschränkt, liegen α_R und α_C ungefähr bei $\pm(10^{-5}..10^{-4})\cdot 1/K$ – Metallfilmwiderständen eher bei $\pm 10^{-5}\cdot 1/K$ und die beiden Kondensatorentypen eher bei $\pm 10^{-4}\cdot 1/K$. Folglich weisen die relativen Temperaturabweichungen $\Delta R/R_{20}$ und $\Delta C/C_{20}$, in dem typischen Betriebstemperaturbereich von -20..80°C, Werte zwischen ±(0,04..0,6) % auf. Da eine gegenseitige Kompensation zwar möglich, aber nicht voraussetzbar ist, muss der relative Temperaturfehler eines einfachen RC-Oszillators mit ±(0,08..1,2) % angesetzt werden.

„On-Chip“-Oszillatoren

Am günstigsten von allen Oszillatoren sind die so genannten „On-Chip“-Oszillatoren, über die viele „low-cost“ µCs verfügen. Damit können nicht nur Bauteilekosten eingespart, sondern die entsprechenden Pins im Allgemeinen auch für I/O-Funktionen verwendet werden – das ist besonders für die kleinen „low-cost“ µCs mit nur acht Pins interessant. Allerdings ist die Genauigkeit von „On-Chip“-Os-

zillatoren die Schlechteste von allen Oszillatoren. Dies soll an einem typischen Beispiel, der „low-cost“ µC-Familie PIC12Cx von Mircochip [121], gezeigt werden. Als Nominalfrequenz der PIC12Cx-„On-Chip“-Oszillatoren ist ein Wert von 4 MHz definiert[131]. Wegen der großen Toleranz wird jeder „On-Chip“-Oszillator werksseitig vermessen und ein Kalibrierwert im Speicher abgelegt. Sofern der Kalibrierwert in das entsprechende Register geschrieben wird, liegt die Oszillatorfrequenz irgendwo zwischen 3,65 und 4,28 MHz, bei 25°C und 5 V Betriebsspannung. Daraus ergibt sich ein relativer Abgleichfehler von -8,75..+7 %. Der Temperaturgang ist nicht linear und im gesamten Betriebstemperaturbereich (-40..+125°C) monoton fallend. Zwischen der unteren und oberen Temperatur nimmt die Frequenz um ca. 300 kHz ab. Folglich bewegt sich der relative Temperaturfehler ungefähr in dem Intervall ±3,75 %. Außerdem weist die Frequenz des „On-Chip“-Oszillators einen Betriebsspannungsdurchgriff von etwa -25 kHz/V auf, weshalb eine gut stabilisierte Betriebsspannung ratsam ist. Unter dieser Annahme spielt der Betriebsspannungsdurchgriff jedoch keine Rolle, so dass der relative Gesamtfrequenzfehler vereinfacht bis zu ca. ±12 % betragen kann[132].

8.5 ASI- contra Standardimpulsspektrum

Der ASI-Bus ist der einzige Bus, der keine rechteckförmigen Standardimpulse mit einer Basisbandübertragung verwendet. Stattdessen arbeitet der ASI-Bus mit sin²-Impulsen. Zum Vergleich sind die Betragsspektren eines Rechteckimpulses und eines sin²-Impulses in Bild 8.2 aufgetragen. Der Rechteckimpuls und das bekanntermaßen korrespondierende si-Spektrum sind gestrichelt gezeichnet. Hingegen betreffen die durchgezogenen Linien den sin²-Impuls und sein Spektrum. Außerdem wurde die Formel für das Spektrum des sin²-Impulses berechnet und so niedergeschrieben, dass die Unterschiede zum Spektrum des Rechteckimpulses deutlich werden.

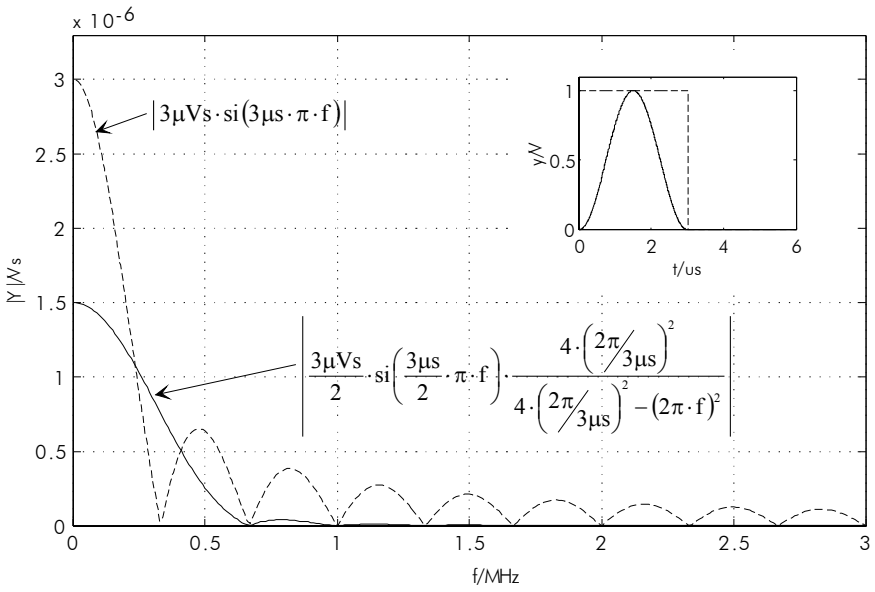


Bild 8.2: Spektren eines sin²-Impulses (durchgezogen) und eines Rechteckimpulses (gestrichelt)

[131] In den µC-Handbüchern werden keine Aussagen oder nur Andeutungen zum Oszillatortyp gemacht. Es dürfte sich jedoch um RC- oder IC-Oszillatoren (I: Konstantstrom) handeln. Weil Stromquellen einfacher zu integrieren sind als Widerstände, sind IC-Oszillatoren wahrscheinlicher. Ein paar der wenigen Andeutungen über den Oszillatortyp in den Handbüchern unterlegen diese Vermutung (z.B. [122]).

[132] Ein unsynchronisierter LIN-Busknoten darf z.B. um ±15 % abweichen [52].

Beim Betrachten der Graphen fällt sofort das viel schnellere Abklingen des sin²-Impulsspektrums mit steigender Frequenz auf. Der Grund hierfür liegt im Nenner der Spektrumsfunktionen. Im Gegensatz zum Spektum des Rechteckimpulses, dessen Hüllkurve mit 1/f abnimmt, geschieht dasselbe beim Spektum des sin²-Impulses mit $1/f^3$. Wie man Bild 8.2 entnehmen kann, ist Letzteres dadurch praktisch ab der ersten Nullstelle bandbegrenzt. Die Nullstellen befinden sich bei $f = 2 \cdot n/T_P$, mit der Impulsdauer $T_p = 3$ µs und der Laufvariablen n ϵ N\0. Für das Spektrum des Rechteckimpulses gelten die Nullstellen $f = n/T_P$, mit ebenfalls der Impulsdauer $T_p = 3$ µs und der Laufvariablen n ϵ N\0. Hieraus erkennt man, dass jede zweite Nullstelle des Rechteckimpulsspektrums mit einer Nullstelle des sin²-Impulsspektrums zusammenfällt. Aus diesem Grund hat das Rechteckimpulsspektrum seine erste Nullstelle früher, wodurch es für niedrige Frequenzen partiell unter dem sin²-Impulsspektrum liegt. Für die EMV spielen jedoch vor allem die höheren Frequenzen eine Rolle. Wegen der Stärkeren Hüllkurvendämpfung schneidet hier das sin²-Impulsspektrum deutlich besser ab als das Rechteckimpulsspektrum.

8.6 SAE-Klassen für Kfz-Netzwerke

Aufgrund stark unterschiedlicher Anforderungen in einem Kfz hat die „Society of Automotive Engineers“ (SAE) die drei folgenden Anwendungs- und Geschwindigkeitsklassen für die Kfz-Vernetzung definiert [3]:

Klasse	Baudrate	Anwendungsdomäne
A	< 10kBits/s	Komfortausstattung (Fensterheber, Sitzheizung, Radio, ...)
B	10 ... 125kBit/s	Allgemeine Datenübertragung (Tacho, Boardcomputer, Kühlwasser ...)
C	> 125kBit/s	Realzeitanwendungen (ESP, ABS, „X-by wire“).

Tabelle 8.3: SAE-Klassen für Kfz-Netzwerke

8.7 Datenintegritätsklassen nach DIN 19244

Die Norm DIN 19244 definiert für Übertragungssysteme die folgenden drei Klassen für die Integrität der Daten [43]:

Klasse	Anwendungen	Beispiele
I1	zyklisch aufdatende Systeme	Fernmessen
I2	spontane Übertragungen	Fernanzeigen, Fernzählen
I3	kritische Informationsübertragungen	Fernsteuern

Tabelle 8.4: Datenintegritätsklassen nach DIN 19244

Ein Übertragungssystem erfüllt die Anforderungen einer Klasse, wenn seine Restfehlerwahrscheinlichkeit bei Variation der mittleren Bitfehlerwahrscheinlichkeit unterhalb der entsprechenden Grenzkurve in Bild 8.3 liegt. Zudem wird in der niedrigsten Klasse I1 eine zyklische Aktualisierung der Daten (Idempotenz) gefordert.

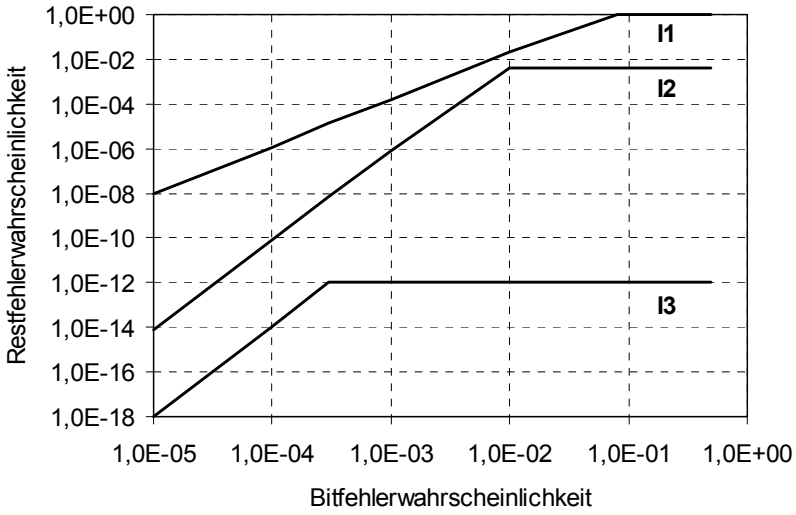


Bild 8.3: Bereiche der Datenintegritätsklassen nach DIN 19244

8.8 Sicherheitskategorien nach EN 954-1

Die Norm EN 954-1 legt in fünf Kategorien das Verhalten von sicherheitsbezogenen Teilen einer Maschinensteuerung im Fehlerfall fest. Hierbei gilt der Grundsatz: Je mehr Risikoverminderung von sicherheitsbezogenen Teilen einer Steuerung abhängen, desto höher sind die Anforderungen an das Systemverhalten bei auftretenden Fehlern. Eine Kurzfassung der Norm enthält Tabelle 8.5, wobei die niedrigste Kategorie die Bezeichnung B (Basiskategorie) trägt [99].

Kategorie	Kurzfassung der Anforderungen	Systemverhalten	Prinzip
B	Die Steuerung muss so konzipiert sein, dass sie den zu erwartenden Einflüssen standhalten kann.	Ein Fehler kann zum Verlust der Sicherheitsfunktion führen.	Überwiegend durch die Auswahl von Bauteilen charakterisiert.
1	Anforderung von B muss erfüllt sein; Verwendung von sicherheitstechnisch bewährten Bauteilen und Prinzipien.	Wie Systemverhalten von B, jedoch mit höherer sicherheitsbezogener Zuverlässigkeit.	
2	Anforderung von 1 muss erfüllt sein; zusätzliche Prüfung der Sicherheitsfunktion in geeigneten Zeitabständen.	Ein aufgetretener Fehler wird erst bei der nächsten Prüfung erkannt.	Überwiegend durch die Struktur der Steuerung charakterisiert.
3	Anforderung von 1 muss erfüllt sein; ein einzelner Fehler darf nicht zum Verlust der Sicherheitsfunktion führen	Die Sicherheitsfunktion bleibt beim Auftreten einzelner Fehler immer erhalten	
4	Anforderung von 3 muss erfüllt sein; der einzelne Fehler muss vor oder bei der nächsten Aufforderung an die Sicherheitsfunktion erkannt werden.	Wenn Fehler auftreten, bleibt die Sicherheitsfunktion erhalten. Die Fehler werden erkannt.	

Tabelle 8.5: Sicherheitskategorien nach EN 954-1

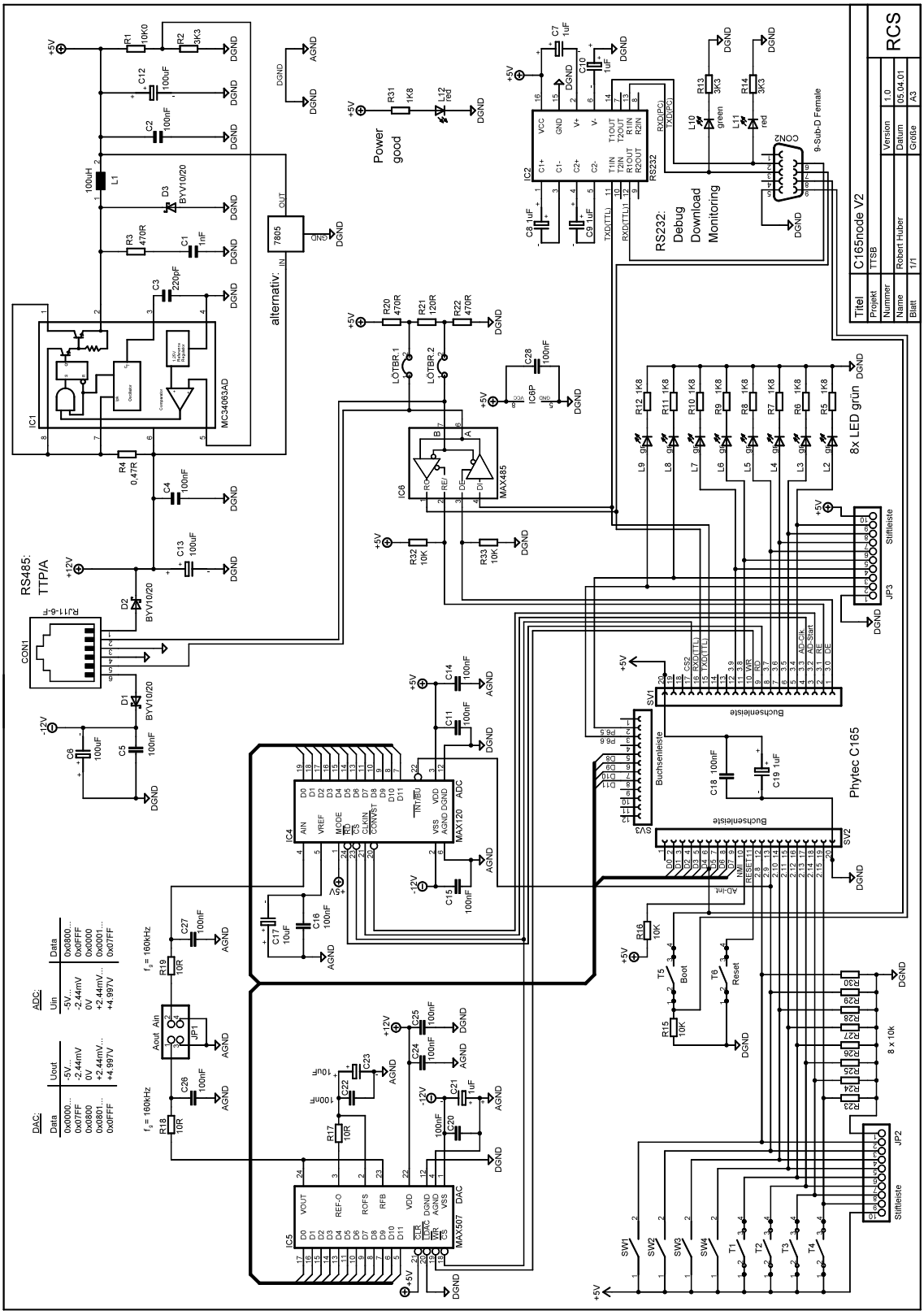
8.9 Potenzielle Standardbausteine

Typ	Bezeichnung	Hersteller	UART	Baudrate	Pins	Kommentar
UART + RS485	MAX3140	Maxim	1	bis 10 MBit/s	28	UART und RS485-Transceiver; Verpolungsschutz durch Phasensteuerung
µC	C16x	Infineon	max. 2	bis 625 kBit/s	100	16 Bit CPU; UART, viele „Timer“ („auto-reload“); ADC, PWM, CAN; viele I/Os, bis 2 KB int. RAM, bis 4 KB int. ROM; „Master“ oder leistungsstarker „Slave“
µC	C166S V2	Infineon	1*	bis 12,5 MBit/s*	100*	C16x-Nachfolger mit 20-fach höherer Leistung und DSP-Funktionen; momentan IP-Core [115] [116] [132]; „Master“
µC	ST10*x*16*x*	ST	max. 2	bis 625 kBit/s	144	wie C16x; jedoch mit internem FLASH; „Master“ oder leistungsstarker „Slave“
µC	ST62*x*18	ST	1**	bis 76,8 kBit/s	20	8 Bit „low-cost” µC; UART, „Auto-Reload-Timer”, ADC, 12 I/Os; „On-Chip”-Oszillator, 192 Bytes RAM, ca. 8 KB EPROM; „Slave”
PSoC***	CY8C25*xxx* CY8C26*xx*	Cypress	bis 8**	bis 6 MBit/s	8 ... 48	8 Bit CPU; UART, „Timer”, Dezimierer, ADC, PWM, CRC, „On-Chip”-Oszillator; bis 256 Bytes RAM, bis 16 KB FLASH; „Slave”
µC	87LPC767	Philips	1	bis 3,3 MBit/s	20	8 Bit „low-cost“µC; UART, 16 Bit „Auto-Reload-Timer“, ADC, I²C, 4 KB EPROM, 128 Bytes RAM, „On-Chip”-Oszillator, bis 18 „I/O”-Pins; „Slave”
µC	AT90S2313	Atmel	1	bis 625 kBit/s	20	8 Bit „low-cost“µC; UART, 16 Bit „Timer“, PWM, Komperator, ca. 2 KB FLASH, 128 Bytes RAM, bis 15 „I/O”-Pins; „Slave”
µC	PIC16F628	Microchip	1	bis 1,25 MBit/s	18	8 Bit „low-cost“µC; UART, 16 Bit „Timer“, PWM, Komperator, ca. 2 KB FLASH, 128 Bytes RAM, „On-Chip”-Oszillator, bis 15 „I/O”-Pins; „Slave”
µC	PIC16F84	Microchip	Software	bis ca. 10 kBit/s*	18	8 Bit „low-cost“µC; 8 Bit „Timer“, , ca. 1 KB FLASH, 68 Bytes RAM, bis 13 „I/O”-Pins; „ultra low-cost Slave”
µC	68HC05	Motorola	1	bis 131 kBit/s	52	8 Bit „low-cost“ µC; UART, 16 Bit CC-„Timer”, ADC, PWM, „On-Chip”-Oszillator, 176B RAM, bis 32KB EPROM, viele I/Os; „Slave”
µC	Z86C08	Zilog	Software	bis ca. 20 kBit/s*	18	8 Bit „low-cost“µC; 8 Bit „Timer“, Komperator, „On-Chip”-Oszillator, 2 KB FLASH, 125 Bytes RAM, bis 14 „I/O”-Pins; „ultra low-cost Slave”
µC	µPD78F9076	NEC	1	bis 2,5 MBit/s	30	8 Bit „low-cost“ µC; UART, 16 Bit „Timer“, 8 Bit-Multipl.izierer, 256 Bytes RAM, 16 KB FLASH, 24 „I/Os”; „Slave“

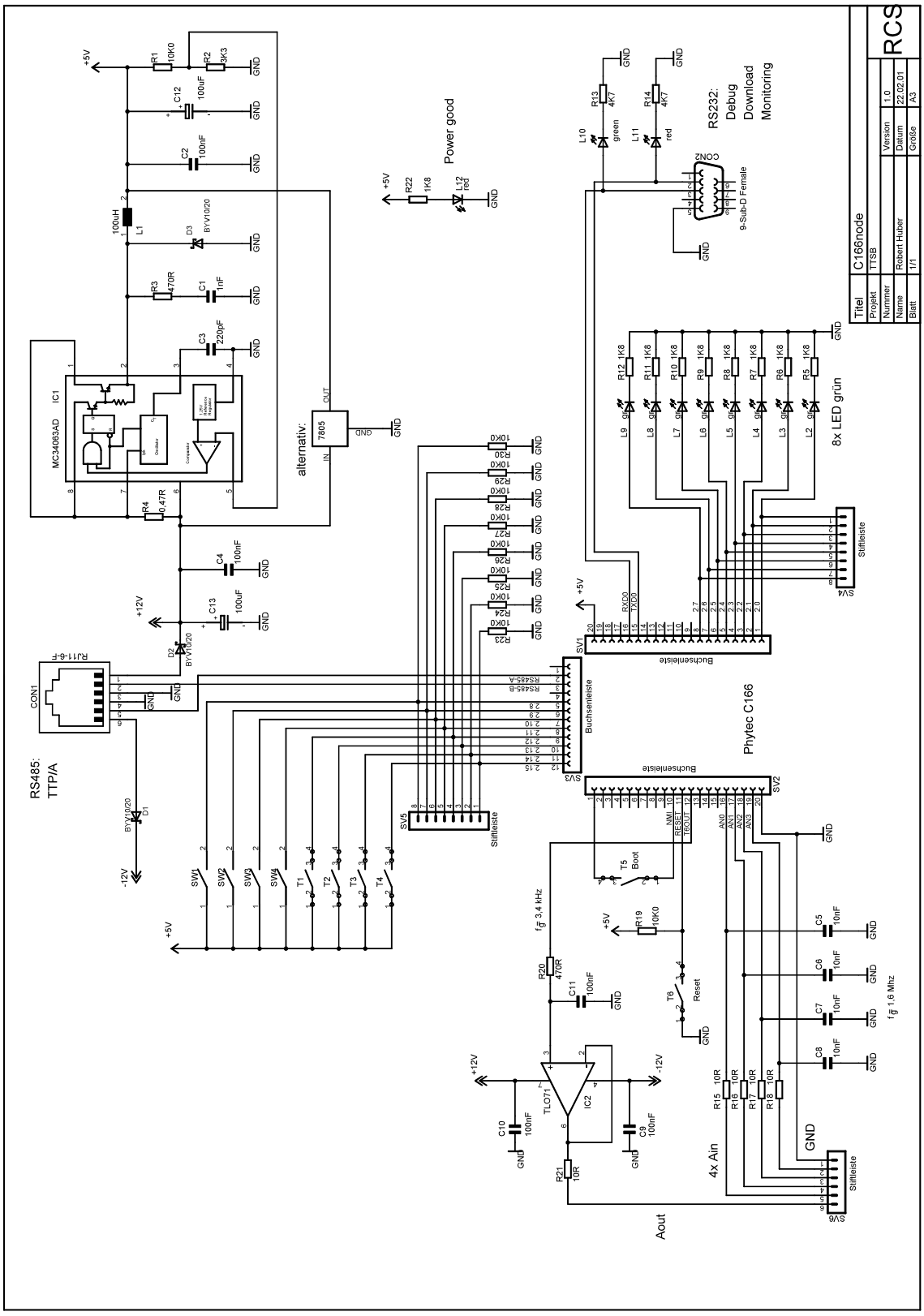
* geschätzter bzw. abgeleiteter Wert, weil keine genaueren Angaben vorlagen
** UART mit 8-fach Abtastung
*** „Programmable-System-on-Chip“; CPU mit konfigurierbarer Peripherie; sehr flexibel; neu

Tabelle 8.6: Potenzielle Standardbausteine für TTP/A

8.10 Schaltplan zum C165node II



8.11 Schaltplan zum C166node



8.12 Schwebekörperversuch

Versuchsaufbau

Der physikalische Aufbau des Schwebekörperversuchs ist in Bild 8.4 abgelichtet. Er wurde freundlicherweise vom Lehrstuhl für Steuerungs- und Regelungstechnik der TU München angefertigt, wofür der Autor Herrn Dr. Freyberger nochmals danken möchte. Ungefähr in Bildmitte ist der rotationssymmetrische, ferromagnetische Schwebekörper zu sehen. Seine Masse beläuft sich auf ca. 160 g. Darüber befindet sich ein, auf vier Alupfosten stehender, Elektromagnet mit Zwangsbelüftung. Eine Lichtschranke dient zum Messen des Abstands zwischen dem Schwebekörper und dem Elektromagnet. Der Messbereich beträgt in etwa 0..20 mm. Zur Lichtschranke gehören die Konstantlichtquelle (rechts) und das Photoelement (links). Das Messprinzip beruht auf der Abschattung des Photoelements durch den Schwebekörper. Je geringer sein Abstand zum Elektromagnet ist, desto weniger Licht trifft auf das Photoelement. Lichtstärke und Photostrom hängen in guter Näherung linear zusammen, so dass an dem ohmschen Abschlusswiderstand des Photoelements eine nahezu abstandsproportionale Sensorspannung entsteht.

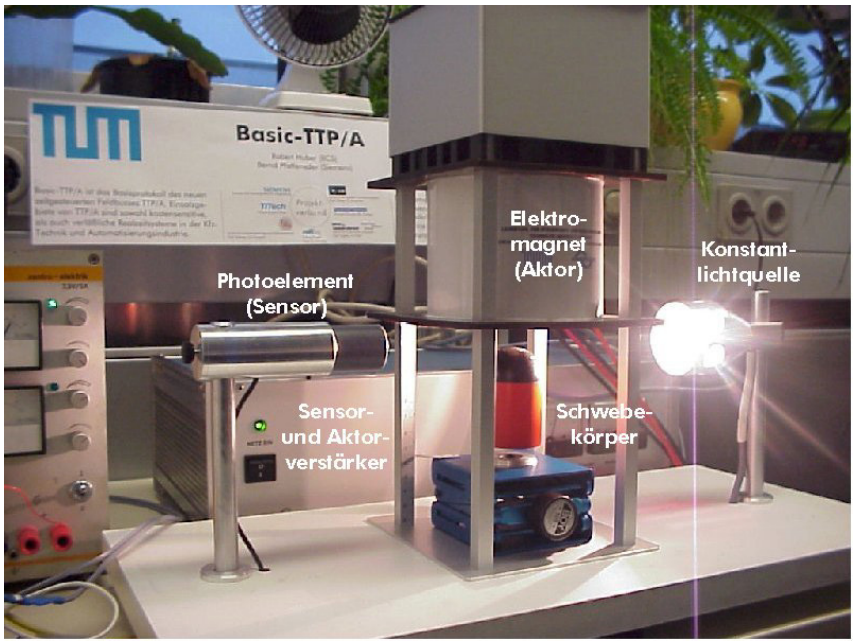


Bild 8.4: Aufbau des Schwebekörperversuchs

Weil die Sensorspannung klein und hochohmig ist, wird sie auf den Bereich -5..+5 V niederohmig abgebildet, bevor sie als Istwert im Regelkreis zur Verfügung steht. Der erforderliche Sensorverstärker ist in dem 19“-Gehäuse hinter dem Schwebekörper eingebaut. Als Stellgröße dient ebenfalls eine Spannung zwischen -5..+5 V. Sie wird einem Aktorverstärker zugeführt, der daraus einen linear abhängigen Strom von -3..+3 A macht und sich im gleichen Gehäuse wie der Sensorverstärker befindet. Dieser Strom fließt durch eine Zylinderspule mit 1000 Windungen und erzeugt ein Magnetfeld, das über einen offenen, stabförmigen Eisenkern mit ca. 140 mm^2 Querschnittsfläche verstärkt wird. Ferner sorgen eine zweite Spule um den Eisenkern und eine Konstantgleichstromquelle in dem 19“-Gehäuse für eine Vormagnetisierung. Dadurch wird zum einen der Aussteuerbereich vergrößert und zum ande-

ren, wegen $R\cdot(i_1 + i_2)^2 > R\cdot i_1^2 + R\cdot i_2^2$, sowohl die Verlustleistung in den Verstärkern als auch die im Elektromagnet verkleinert. Prinzipiell notwendig ist die Vormagnetisierung allerdings nicht. Die Höhe des Konstantgleichstroms beträgt umgerechnet etwa 1 A – die Windungszahlen der Spulen sind nicht gleich – und ist so gewählt, dass der Elektromagnet ohne Steuerstrom den Schwebekörper gerade nicht festhält. Selbstverständlich kann man die Vormagnetisierung auch stärker auslegen. Dann kann es jedoch passieren, dass der Schwebekörper beim Abschalten der Regelung nach oben, anstatt nach unten fällt.

Für die Magnetkraft ist die Gesetzmäßigkeit $dF = B^2/2\cdot(1/\mu_1-1/\mu_2)\cdot dA$ verantwortlich. Sie besagt, dass die Kraft dF in Richtung der kleineren Permeabilität μ_x auf eine senkrechte Grenzfläche dA in einem Magnetfeld B wirkt. Aufgrund von $\mu_{FE} >> \mu_{Luft}$, treten die Feldlinie auf der gesamten Oberfläche des Schwebekörpers nahezu senkrecht ein oder aus. Folglich wirken rundum den Schwebekörper nahezu senkrecht ziehende Kräfte der Größe $dF \approx B^2/(2\cdot\mu_{Luft})$. Des Weiteren bewirken das inhomogene äußere Spulenfeld und die Feldverzerrungen durch den ferromagnetischen Schwebekörper ein stark inhomogenes Magnetfeld.. Würde man sich auf der Schwebekörperoberfläche zum Elektromagnet hin bewegen, so könnte man eine stetige Zunahme der Magnetfeldstärke beobachten ($B_{oben} > B_{unten}$). Aus diesem Grund sind die Magnetkräfte auf der Oberseite des Schwebekörpers stärker als auf der Unterseite ($F_{oben} > F_{unten}$). Mit der resultierenden Kraft zieht der Elektromagnet den Schwebekörper an. Und zwar nicht nur nach oben, sondern auch seitlich, wenn sich der Schwebekörper azentrisch unter dem Elektromagnet befindet. In diesem Fall konzentriert sich das Magnetfeld auf der Seite des Schwebekörpers, wo der Elektromagnet ist. Daraus resultiert eine Seitenkraft in Richtung Spulenmitten, wodurch eine selbstständige Zentrierung entsteht. Sofern der Schwebekörper rotationssymmetrisch mit einer kugelförmigen Kappe oder gar eine Kugel ist, wird seine Ruhelage stets mittig unter dem Elektromagneten sein.

Regelkreis

Damit der Schwebekörper trotz Störungen seine Position hält, benötigt man eine Regelung. Ein Standardregelkreis, so wie in Bild 8.5 dargestellt und im Versuchsaufbau eingesetzt, reicht hierfür vollends aus. Der Regler muss aus Stabilitätsgründen vom Typ PD oder PID sein – eine Stabilitätsuntersuchung folgt später. Mit dem PD-Regler hat der Regelkreis ein besseres Führungsverhalten und mit dem PID-Regler ein besseres Lageverhalten. Da der Regler mit digitalen Mitteln realisiert ist (siehe Abschnitt 5.4), enthält der Regelkreis einen A/D- und D/A-Wandler. Die Auflösungen betragen je 12 Bit und die Abtastfrequenz f_A beträgt ca. 1 kHz. Beide Werte dürfen auch kleiner gewählt werden, weil der Regelkreis sehr robust ist. Mit z.B. einer 10 Bit-Auflösung oder einer Abtastperiode von 5 ms funktioniert das Ganze immer noch zufriedenstellend. Sinn und Zweck des Demonetzes war es jedoch die Leistungsfähigkeit von „Basic-TTP/A“ unter Beweis zu stellen, weshalb insbesondere die Abtastfrequenz hoch gewählt wurde. Aufgrund der Abtastung entsteht eine durchschnittliche Totzeit von $1/(2\cdot f_A) \approx 500$ µs. Sie wird durch die Übertragung der Regeldifferenz um ca. 300 µs vergrößert, weil der Regelkreis über „Basic-TTP/A“ geschlossen ist (siehe Abschnitt 5.4.2). Unter Vernachlässigung der relativ kurzen Rechenzeit, beträgt die Gesamttotzeit also ca. 800 µs. Dank der erwähnten Robustheit, wirkt sie sich weder merklich auf die Stabilität, noch auf die Güte der Regelung aus. Man kann das z.B. mit Hilfe des Matlab/Simulink-Modells auf der beiliegenden CD simulatorisch nachvollziehen.

Als Folge der Vormagnetisierung muss der minimale Ausgangsstrom des Aktorverstärkers begrenzt werden. Wie Bild 8.5 zeigt, kompensiert er die Vormagnetisierung, so dass der faktische Stellbereich des Steuerstroms 0..4 A beträgt. Ohne die Begrenzung besteht die Gefahr einer Instabilität, aufgrund der nichtlinearen Regelstrecke. Es sei angenommen, dass sich der Schwebekörper, z.B. wegen einer Störung, aus seiner Ruhelage zum Elektromagnet hin bewegt. Daraufhin senkt der Regler den Aus-

gangsstrom, um die Magnetkraft zu verringern. Je nach Regeldifferenz, Reglertyp und Störddauer, kann der Ausgangsstrom auch negativ werden. Sollte er die Vormagnetisierung überkompensieren, findet eine Umpolung des Magnetfeldes statt. Dadurch entsteht aus der Gegenkopplung eine Mitkopplung, weil das umgepolte Magnetfeld den Schwebekörper nicht abstößt, sondern ebenso anzieht. Überwindet die Kraft des umgepolten Magnetfeldes die Gravitationskraft, dann schnellt der Schwebekörper zum Elektromagnet und verharrt dort. Ein linearer Regler findet aus dieser „Deadlock“ bzw. aus diesem instabilen Zustand nicht selbstständig heraus. Anstatt den Ausgangsstrom positiver zu machen, damit die Gravitation den Schwebekörper nach unten bewegen kann, macht ein linearer Regler bis zur Sättigung das Gegenteil, wodurch der Schwebekörper immer fester gehalten wird.

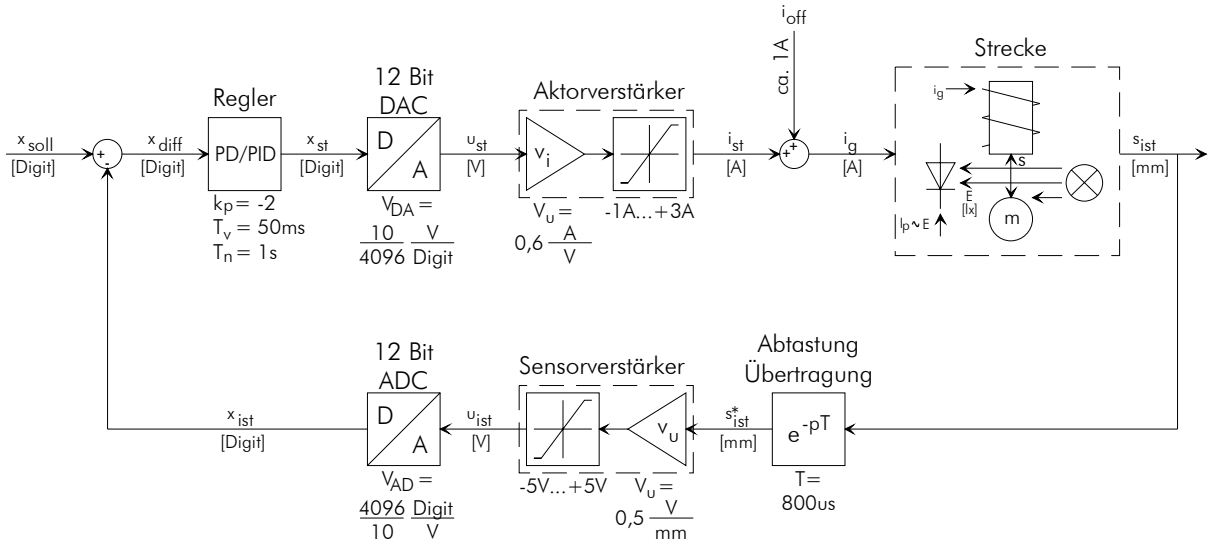
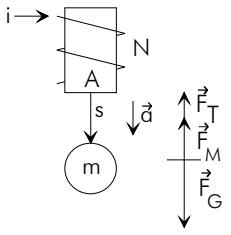


Bild 8.5: Regelkreis des Schwebekörperversuchs

Regelstrecke

Für eine systemtheoretische Analyse des Regelkreises ist eine mathematische Modellierung der Regelstrecke (kurz Strecke) notwendig. Dazu wird eine Kräftebilanz am Schwebekörper aufgestellt. Gemäß Bild 8.6, zieht die Gewichtskraft $F_G = m \cdot g$ den Schwebekörper nach unten, währenddessen ihn die Magnetkraft $F_M \approx ½ \cdot \mu_0 \cdot A \cdot (N \cdot i/s)^2$ nach oben zieht. Wenn sich beide Kräfte die Waage halten, also im stationären Fall, schwebt der Körper. Befinden sich die Kräfte dagegen im Ungleichgewicht, wird der Körper mit und in Richtung der resultierenden Kraft beschleunigt. Folglich muss in der Kräftebilanz zusätzlich die Massenträgheitskraft $F_T = m \cdot a$ berücksichtigt werden, um auch den dynamischen Fall abzudecken.



$$\sum \vec{F} = 0 \rightarrow \vec{F}_M + \vec{F}_G + \vec{F}_T = 0 \rightarrow F_M - F_G + F_T = 0$$

$$F_M \approx \tfrac{1}{2} \cdot \mu_0 \cdot A \cdot \left(\frac{N \cdot i}{s}\right)^2 \qquad \mu_0 = 4 \cdot \pi \cdot 10^{-7} \frac{Vs}{Am} \qquad A = 140 mm^2$$

$$\vec{F}_G = m \cdot \vec{g} \qquad F_G = m \cdot g \qquad N = 1000 \qquad g = 9{,}81 \frac{m}{s^2}$$

$$\vec{F}_T = -m \cdot \vec{a} \qquad F_T = m \cdot a = m \cdot \frac{d^2 s}{dt^2} \qquad m = 160\,g$$

Bild 8.6: Kräftegleichgewicht am Schwebekörper

Da die Massenträgheitskraft F_T, je nach Gewichts- und Magnetkraft, ihre Richtung wechseln kann, benötigt man ein Bezugssystem. Es wurde in Bild 8.6 so festgelegt, dass der Weg s nach unten hin wächst, also mit dem Abstand des Schwebekörpers zum Elektromagnet. Demnach zeigt eine positive Beschleunigung a nach unten und eine positive Massenträgheitskraft F_T nach oben. Zwar widerspricht diese Zählrichtung dem allgemeinen Empfinden, weil Tiefe zumeist mit einer negativen Zahl verbunden wird, sie macht jedoch die Modellierung einfacher. Würde man s entgegengesetzt zählen, hätte das hauptsächlich auf die Formel für die Magnetkraft F_M ungünstige Auswirkungen. Anstelle von s^2, stünde dann im Nenner $(s_0 - s^*)^2$, wodurch die weiteren Berechnungen nur umständlicher ausfielen.

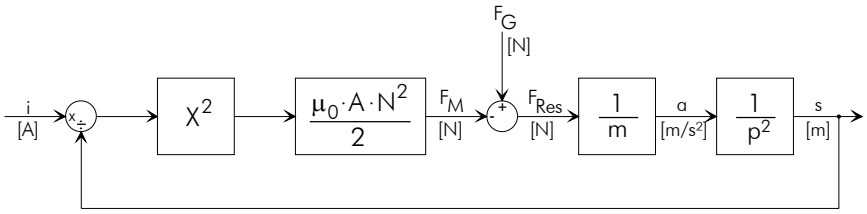


Bild 8.7: Signalflussplan der Strecke

Aus dem Kräftegleichgewicht, den Formeln für die Kräfte und der Beziehung $F_{Res} = -F_T$ wurde in Bild 8.7 ein Signalflussplan aufgestellt. Er gibt den physikalischen Sachverhalt in systemtheoretischer Form wider und verschafft tiefe Einblicke in die Regelstrecke. Die Eingangsgröße ist der Magnetisierungsstrom i und die Ausgangsgröße der Abstand bzw. Weg s. Aufgrund des doppelten Integrators, welcher die Strecke strukturinstabil macht, benötigt man einen Regler mit D-Anteil. Ins Unreine gesprochen, dreht der Doppelintegrator die Phase um $-\pi$ und der Differenzierer um $+½\cdot\pi$, so dass ein stabiler Phasenrand einstellbar ist. Diese Betrachtungsweise ist deshalb nicht ganz korrekt, weil die Strecke erstens nichtlinear ist und zweitens eine Rückkopplung besitzt. Dadurch tritt der Doppelintegrator nach außen hin etwas anders in Erscheinung. Allerdings spiegelt die obige Betrachtungsweise den Sachverhalt für eine prägnante Erklärung ausreichend gut wider.

Eine Übertragungsfunktion lässt sich für die Strecke nicht so ohne Weiteres aufstellen. Der Grund ist die nichtlineare Abhängigkeit der magnetischen Kraft F_M vom Strom i und Weg s. Für eine standardmäßige Stabilitätsuntersuchung ist jedoch eine Übertragungsfunktion erforderlich. Zu diesem Zweck wird die Strecke im Arbeitspunkt (i_0, s_0) linearisiert und eine Übertragungsfunktion für den Kleinsignalbetrieb abgeleitet. Die wichtigsten Rechenschritte sind nachfolgend aufgeführt:

$$F_M \approx \frac{1}{2}\cdot\mu_0\cdot A\cdot\left(\frac{N\cdot i}{s}\right)^2 \approx F_M(i_0,s_0) + \left.\frac{\partial F_M(i,s)}{\partial i}\right|_{i_0,s_0}\cdot\Delta i + \left.\frac{\partial F_M(i,s)}{\partial s}\right|_{i_0,s_0}\cdot\Delta s$$

$$\frac{\partial F_M(i,s)}{\partial i} = \mu_0\cdot A\cdot N^2\cdot\frac{i}{s^2} \qquad \frac{\partial F_M(i,s)}{\partial s} = -\mu_0\cdot A\cdot N^2\cdot\frac{i^2}{s^3}$$

$$F_M - F_G + F_T = 0 \;\Rightarrow\; \frac{1}{2}\cdot c_1\cdot\left(\frac{i_0}{s_0}\right)^2 + c_1\cdot\frac{i_0}{s_0^2}\cdot\Delta i - c_1\cdot\frac{i_0^2}{s_0^3}\cdot\Delta s - m\cdot g + m\cdot\frac{d^2(s_0+\Delta s)}{dt^2} \approx 0$$

$$\text{mit}\quad c_1 = \mu_0\cdot A\cdot N^2 \approx 1{,}76\cdot10^{-4}\,\frac{m^2}{As^2}\quad\text{und}\quad s = s_0+\Delta s\quad\text{und}\quad i = i_0+\Delta i$$

Die linearisierte Differenzialgleichung enthält einen statischen und einen dynamischen Teil. Der statische Teil beschreibt den Arbeitspunkt (i_0, s_0) und ist in Formel 8.4 separat dargestellt.

$$\frac{1}{2}\cdot c_1\cdot\left(\frac{i_0}{s_0}\right)^2 - m\cdot g + \frac{d^2 s_0}{dt^2} = 0 \qquad \text{wobei } \frac{d^2 s_0}{dt^2} = 0$$

$$\Rightarrow \qquad i_0 = s_0\cdot\sqrt{\frac{2\cdot m\cdot g}{c_1}}$$

Formel 8.4: Arbeitspunktgleichung der Strecke

Von größerer Bedeutung für die Stabilitätsuntersuchung ist der dynamische Teil. Er beschreibt das zeitliche Verhalten der Strecke für kleine Änderungen (Δi, Δs) um den Arbeitspunkt (i_0, s_0) und wurde ebenfalls separiert. Die entsprechende Differenzialgleichung ist in Formel 8.5 zu sehen, wobei i_0 mittels der algebraischen Arbeitspunktgleichung eliminiert wurde.

$$c_1\cdot\frac{i_0}{s_0^2}\cdot\Delta i - c_1\cdot\frac{i_0^2}{s_0^3}\cdot\Delta s + m\cdot\frac{d^2\Delta s}{dt^2}\approx 0$$

$$\Rightarrow \qquad \frac{1}{s_0}\cdot\sqrt{\frac{c_1\cdot 2\cdot g}{m}}\cdot\Delta i \approx \frac{2\cdot g}{s_0}\cdot\Delta s - \frac{d^2\Delta s}{dt^2}$$

Formel 8.5: Differenzialgleichung der Strecke für kleine Änderungen

Aus dieser Differenzialgleichung lässt sich nun die gesuchte Übertragungsfunktion bilden. Dazu wird die Differenzialgleichung einer Laplace-Transformation unterzogen und die resultierende Gleichung entsprechend Formel 8.6 umgestellt. Die erwähnte Instabilität zeigt sich in dem positiven Pol p_1 der Übertragungsfunktion F_S.

$$\Delta i(t)\rightarrow I(p) \qquad \Delta s(t)\rightarrow S(p) \qquad \frac{d^2}{dt^2}\rightarrow p^2$$

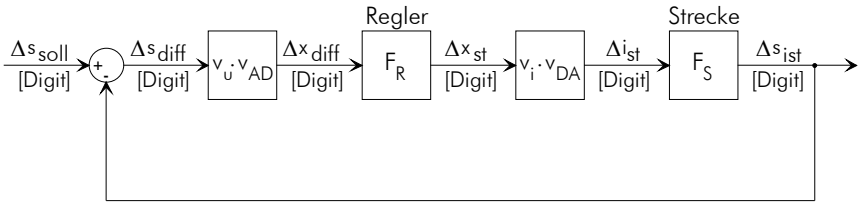
$$\frac{1}{s_0}\cdot\sqrt{\frac{c_1\cdot 2\cdot g}{m}}\cdot I \approx \frac{2\cdot g}{s_0}\cdot S + p^2\cdot S$$

$$\Rightarrow \qquad F_S = \frac{S}{I}\approx \frac{-\frac{1}{s_0}\cdot\sqrt{\frac{c_1\cdot 2\cdot g}{m}}}{p^2 - \frac{2\cdot g}{s_0}} \qquad \text{Pole}: \begin{cases} p_1 = +\sqrt{\frac{2\cdot g}{s_0}} \\ p_2 = -\sqrt{\frac{2\cdot g}{s_0}} \end{cases}$$

Formel 8.6: Übertragungsfunktion der linearisierten Strecke

Stabilität

Die Untersuchung der Stabilität erfolgt an den charakteristischen Polynomen ($1 - F_0$) des linearisierten Regelkreises mit PD- und PID-Regler. Hierbei bleiben die Totzeit, die Begrenzungen, die Abtastung und Quantisierungseffekte unberücksichtigt (siehe Bild 8.5). Wie sich zeigen wird, hat der stabile Regelkreis sehr langsame Pole, so dass die Phasendrehung durch die Totzeit nicht ins Gewicht fällt. Aus dem gleichen Grund darf auch von einem zeitkontinuierlichen Verhalten ausgegangen werden. Des Weiteren spielen die Begrenzungen keine Rolle, weil nur das Kleinsignalverhalten interessiert. Und schließlich sind die Auflösungen des A/D- und D/A-Wandlers so hoch ($\Delta s \approx 5$ µm), dass ein wertekontinuierlicher Ansatz keine nennenswerten Fehler hervorruft. Zur Bildung der charakteristischen Polynome werden die Führungsübertragungsfunktionen des Regelkreises $F_{w,\,PD}$ und $F_{w,\,PID}$ aufgestellt. Das ist zwar prinzipiell nicht nötig, doch bieten sie weitere Einblicke in die Dynamik des Systems. Außerdem wird der Regelkreis äquivalent umgeformt, damit die Ein- und Ausgangsgröße dieselbe Einheit besitzen, wodurch sich die Auswertungen vereinfachen. Da diese Rechenschritte aufwändig, aber nicht sonderlich schwer sind, werden anschließend nur die Ergebnisse präsentiert:



$$F_w = \frac{S_{ist}}{S_{soll}} = \frac{v_u \cdot v_i \cdot v_{AD} \cdot v_{DA} \cdot F_R \cdot F_S}{1 + v_u \cdot v_i \cdot v_{AD} \cdot v_{DA} \cdot F_R \cdot F_S}$$

$$\text{mit } v_{AD} = \frac{1}{v_{DA}} \quad \text{und} \quad F_{R,\,PD} = k_p \cdot (1 + T_V \cdot p) \quad \text{und} \quad F_{R,\,PID} = k_p \cdot \left(1 + T_V \cdot p + \frac{1}{T_n \cdot p}\right)$$

$$\Rightarrow \qquad F_{w,\,PD} = \frac{c_2 \cdot \left(p + \frac{1}{T_v}\right)}{p^2 + c_2 \cdot p + c_3} \qquad\qquad F_{w,\,PID} = \frac{c_2 \cdot \left(p^2 + \frac{p}{T_v} + \frac{1}{T_v \cdot T_n}\right)}{p^3 + c_2 \cdot p^2 + c_3 \cdot p + c_4}$$

$$\text{mit} \qquad c_2 = -k_p \cdot T_v \cdot v_u \cdot v_i \cdot \frac{1}{s_0} \cdot \sqrt{\frac{c_1 \cdot 2 \cdot g}{m}} \approx -44{,}07 \frac{m}{s^2} \cdot \frac{k_p \cdot T_v}{s_0}$$

$$c_3 = -k_p \cdot v_u \cdot v_i \cdot \frac{1}{s_0} \cdot \sqrt{\frac{c_1 \cdot 2 \cdot g}{m}} - \frac{2 \cdot g}{s_0} \approx -44{,}07 \frac{m}{s^2} \cdot \frac{k_p}{s_0} - 19{,}62 \frac{m}{s^2} \cdot \frac{1}{s_0}$$

$$c_4 = -\frac{k_p}{T_n} \cdot v_u \cdot v_i \cdot \frac{1}{s_0} \cdot \sqrt{\frac{c_1 \cdot 2 \cdot g}{m}} \approx -44{,}07 \frac{m}{s^2} \cdot \frac{k_p}{T_n \cdot s_0}$$

Formel 8.7: Führungsübertragungsfunktionen F_w des linearisierten Regelkreises mit PD- und PID-Regler

Für die Beurteilung der Stabilität werden die Wurzelortskurven (WOKs) der beiden charakteristischen Polynome berechnet. Es gilt also die Gleichungen $p^2 + c_2 \cdot p + c_3 = 0$ für einen PD-Regler und $p^3 + c_2 \cdot p^2 + c_3 \cdot p + c_4 = 0$ für einen PID-Regler in Abhängigkeit von den Reglergrößen k_p, T_v, T_n und dem Arbeitspunkt s_0 in p zu lösen. Die Vielzahl der Parameter bereitet jedoch Darstellungsprobleme, weshalb in den nachfolgenden WOKs nur die Wichtigsten variiert werden. Hierzu zählen in erster Linie der Arbeitspunkt s_0 und, aufgrund des notwendigen D-Anteils im Regler, in zweiter Linie die Vorhaltezeit T_v. Für die Verstärkung k_p und die Nachstellzeit T_n werden die konstanten Werte aus Bild 8.5 eingesetzt.

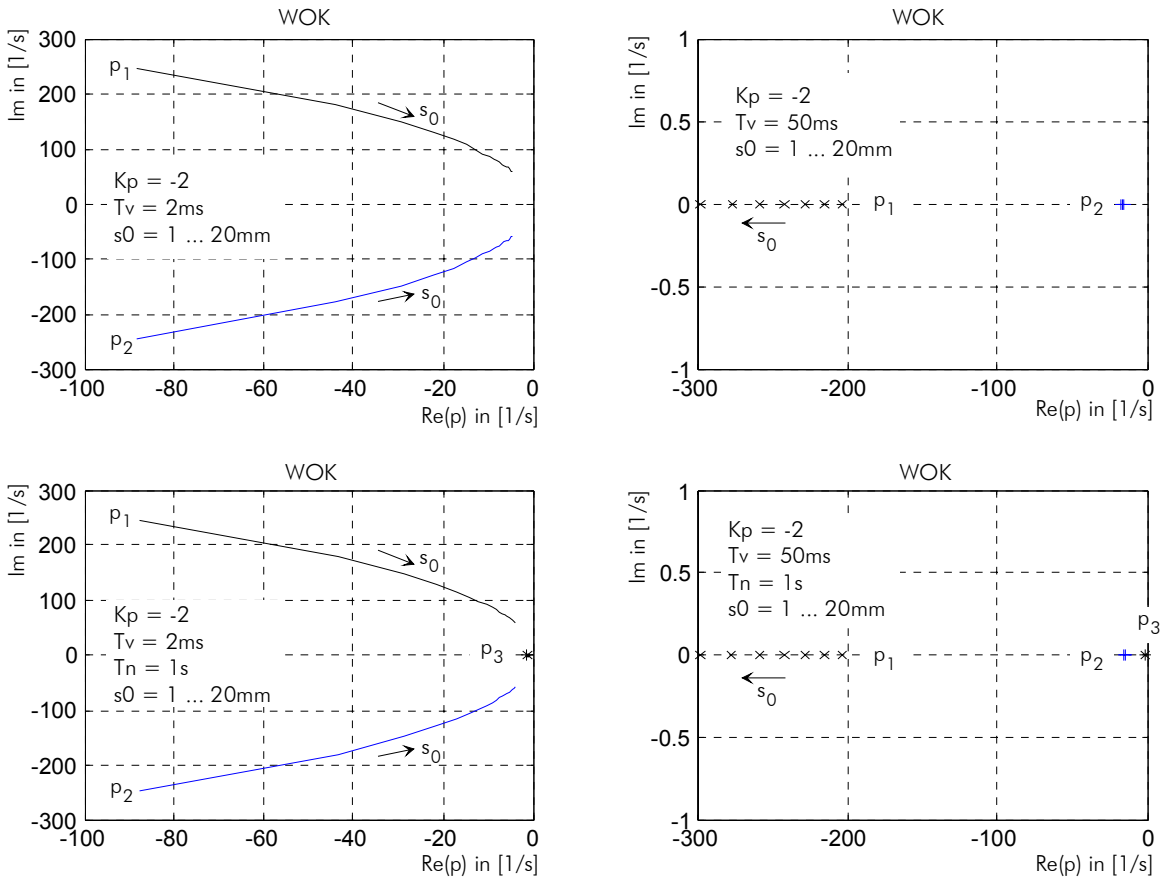


Bild 8.8: WOKs für den Regelkreis mit PD- (oben) und PID-Regler (unten)

Die oberen WOKs gehören zum Regelkreis mit PD-Regler und die unteren WOKs zum Regelkreis mit PID-Regler. Für deren Ähnlichkeit ist die Nachstellzeit T_n verantwortlich. Je größer sie gewählt wird, desto weniger macht sich der Integrator in der Dynamik bemerkbar. Für $T_n \rightarrow \infty$ wären die entsprechend WOKs absolut gleich, weil der PID-Regler dann in einen PD-Regler überginge. In den angegebenen Parametersätzen ist T_n zwar nicht unendlich, aber im Vergleich zu T_v relativ groß. Aus diesem Grund macht sich T_n in den schnelleren Polen p_1 und p_2 kaum bemerkbar. Der Variationsbereich für den Arbeitspunkt s_0 beginnt bei 1 mm, nimmt in Pfeilrichtung zu und endet bei 20 mm. Zudem wurde der D-Anteil in den linken Diagrammen klein ($T_v = 2$ ms) und in den rechten Diagrammen groß ($T_v = 50$ ms) gewählt. Weil in allen WOKs sämtliche Pole in der linken p-Halbebene liegen, ist eine unmittelbare Erkenntnis die Stabilität des Regelkreises für die dargestellten Parametersätze im gesamten Arbeitsbereich. Allerdings zeigt sich an den konjugiert komplexen Polpaaren, dass der Regel-

kreis bei dem kleinen T_v-Wert zum Schwingen neigt ($f \approx (50..250)/2\pi$ Hz). Würde man T_v weiter verringern, so entstünde unterhalb eines Grenzwerts daraus eine Instabilität. Mit einem entsprechend hohen T_v-Wert lässt die Schwingneigung andererseits vollends unterdrücken. Das belegen die beiden rechten WOKs durch ihr überaperiodisches Verhalten, woraus die erwähnte Notwendigkeit eines Reglers mit D-Anteil offensichtlich wird.

Ferner ist eine zunehmende Trägheit des Regelkreises mit wachsendem s_0 zu verzeichnen. Man erkennt das in den beiden linken WOKs, aufgrund des kleineren Maßstabs, besonders gut. Dieser Effekt kommt durch die abstandsabhängige Magnetkraft F_M zustande. Gemäß der Formel aus Bild 8.6, besteht zwischen der Magnetkraft und dem Abstand der funktionale Zusammenhang $F_M \sim 1/s^2$. Folglich nimmt die Magnetkraft und damit die Kreisverstärkung $|F_0|$ mit dem Abstand des Schwebekörpers zum Elektromagneten ab. Je geringer aber die Kreisverstärkung ist, desto weniger Dynamik besitzt der Regelkreis. Für $s \to \infty$ würde sie ganz verwinden, weil die Kreisverstärkung gegen Null geht. Das käme einem offenen Regelkreis gleich und würde eine Instabilität bedeuten, weil der Schwebekörper dann frei nach unten fiele. Aufgrund der Strombegrenzung im Aktorverstärker, tritt dieser Fall in Praxis jedoch schon bei einem Abstand größer ca. 20 mm ein. Darüber hinaus kann man vor allem den WOKs auf rechten Seite in Bild 8.8 entnehmen, dass die Abtast- und Totzeit keinen großen Einfluss auf das Regelkreisverhalten haben. Hierzu soll das Beispiel PD-Regler mit überaperiodischem Verhalten näher betrachten werden. Der Realteil des langsamen Pols p_2 beträgt ungefähr -20 1/s, woraus sich ein Systemzeitkonstante von ca. 50 ms ableiten lässt. Im Vergleich dazu fallen die Abtastzeit mit etwa 1 ms und die Totzeit mit etwa 800 µs sehr gering aus. Sie sind ca. 50..60 mal kleiner und können bereits ab einem Faktor um zehn herum getrost vernachlässigt werden.

Robustheit

Ein weiterer Punkt ist die (Parameter-)Robustheit des Regelkreises. Aus den beiden rechten WOKs in Bild 8.8 geht hervor, dass ein entsprechend großer D-Anteil im Regler erheblich dazu beiträgt. Man sieht das an dem dominanten Pol p_2 (PD-Regler), bzw. den dominanten Polen p_2 und p_3 (PID-Regler). Bei einer Veränderung des Arbeitspunkts s_0 bleibt er, bzw. bleiben sie, nahezu ruhig, währenddessen der Pol p_1 deutlich wandert. Der Pol p_1 hat auf das Systemverhalten jedoch kaum Einfluss, weil er für die viel schnelleren Einschwingvorgänge verantwortlich ist. Folglich hängt das Systemverhalten in den beiden rechten WOKs aus Bild 8.8 nur unmerklich vom Arbeitspunkt s_0 ab – diese Erkenntnis deckt sich mit den praktischen Erfahrungen. Ein ähnlich gutmütiges Verhalten zeigt sich auch beim Variieren der Reglerverstärkung k_p. Der Beweis soll anhand der charakteristischen Polynome aus Formel 8.7 erbracht werden[133]. Für die Lage der Polstellen sind die Koeffizienten c_2 und c_3 bzw. c_4 ausschlaggebend. Sie hängen sowohl vom Arbeitspunkt s_0 als auch von der Reglerverstärkung k_p ab. Bis auf einen additiven Term im Koeffizienten c_3, unterscheiden sich die Wirkungen von k_p und s_0 auf die Koeffizienten nur durch ihre Reziprozität. Deshalb gleichen sich die entsprechenden WOKs in ihrer Form und werden gegensinnig durchlaufen. Fakt bleibt aber, dass mit einem PD-Regler der Pol p_2 in einem großen k_p-Bereich dominant ist und sich kaum bewegt. Im Fall PID-Regler gilt dasselbe für die Pole p_2 und p_3.

Zudem bewirkt ein entsprechend großer D-Anteil im Regler ein sanftes Einschwingen. Wie Bild 8.8 zeigt, liegen sämtliche Pole in den rechten WOKs auf der negativen reellen p-Achse. Das ist für diesen technischen Prozess insofern wichtig, damit der Schwebekörper z.B. bei einer Sollwertänderung nicht am Elektromagneten anschlägt oder den Fangbereich verlässt. Allerdings sollte der D-Anteil, trotz den genannten positiven Auswirkungen, nicht übermäßig groß gemacht werden. Denn er übersteuert schnell den Aktorverstärker und erhöht das Rauschen im Regelkreis. Einen diesbezüglich guten Kom-

[133] Eine Erklärung über die Kreisverstärkung $|F_0|$ ist auch möglich.

promiss stellen die WOKs im rechten Teil von Bild 8.8 dar. Dort wurde der T_v-Wert so gewählt, dass erstens alle Pole auf der reellen p-Achse liegen und zweitens sich ein deutliches, aber nicht zu deutliches, überaperiodisches Regelkreisverhalten einstellt.

Selbstverständlich gibt es weitere Parametersätze, mit denen sich ein ähnliches gutes Verhalten einstellt. Dazu soll der Regelkreis mit PD-Regler noch genauer untersucht werden – mit einem PID-Regler gelten die Ergebnisse ebenso, sofern die Nachstellzeit T_n groß ist. Gemäß Formel 8.7, lautet das charakteristische Polynom für den Regelkreis mit PD-Regler $C(p) = p^2 + c_2 \cdot p + c_3$. Seine Wurzeln berechnen sich wie folgt:

$$p^2 + c_2 \cdot p + c_3 = 0 \quad \Rightarrow \quad p_{1,2} = -\frac{c_2}{2} \pm \sqrt{\left(\frac{c_2}{2}\right)^2 - c_3}\ .$$

Damit der Regelkreis stabil ist, müssen die beiden Wurzel p_1 und p_2 negative Realteile besitzen. Ferner können T_v und s_0 nur positive Werte annehmen, so dass sich für k_p die Restriktion ergibt:

$$1)\ \frac{c_2}{2} > 0 \quad \rightarrow \quad k_p < 0,\ \text{weil}\, T_v > 0\ \text{und}\ s_0 > 0$$

$$2)\ -\frac{c_2}{2} + \sqrt{\left(\frac{c_2}{2}\right)^2 - c_3} < 0 \quad \rightarrow \quad \left(\frac{c_2}{2}\right)^2 - c_3 < \left(\frac{c_2}{2}\right)^2 \quad \rightarrow \quad c_3 > 0$$

$$c_3 \approx -44{,}07 \frac{m}{s^2} \cdot \frac{k_p}{s_0} - 19{,}62 \frac{m}{s^2} \cdot \frac{1}{s_0} > 0$$

$$\Rightarrow \quad k_p < -0{,}445.$$

Formel 8.8: Stabilitätsbedingung für die Reglerverstärkung k_p

Das Einschwingverhalten ist mit $s_0 > 0$, $T_v > 0$ und $k_p < -0{,}445$ allerdings nicht festgelegt. Hierfür muss man die Diskriminante $(c_2/2)^2 - c_3$ näher betrachten. Aus den geschilderten Gründen ist ein sanftes, überaperiodisches Einschwingen wünschenswert. Deshalb muss die Diskriminante größer Null sein. Nach Ersetzen von c_2 und c_3 durch k_p, T_v und s_0 (siehe Formel 8.7) und ein paar Umformungen folgt daraus $k_p^2 \cdot T_v^2 + k_p \cdot s_0 \cdot 11\ s^2/m + s_0 \cdot 24{,}74\ s^2/m > 0$. Da k_p negativ und s_0 positiv sind, kann der mittlere Term diese Bedingung verletzen. Ein Beispiel ist der Parametersatz in den rechten WOKs aus Bild 8.8. Natürlich ist er nicht der Einzige, sondern einer von unendlich vielen. Das Gleiche gilt auch für die Parametersätze, mit denen die Ungleichung erfüllt wird. Für eine Abschätzung sind in Bild 8.9 die Parametersätze für den aperiodischen Fall eingezeichnet. Oberhalb der Grenzfläche liegt das Parametergebiet für den überaperiodischen Fall. Man erkennt zum einen, dass mit der gewählten Reglerverstärkung $k_P = -2$ und Vorhaltezeit $T_V = 50$ ms der gewünschte Abstand zur Grenzfläche im gesamten Arbeitsbereich deutlich, aber nicht übermäßig ist. Zum anderen zeigt Bild 8.9 die Robustheit des Regelkreises. Sofern T_v nicht zu klein ist, können die Parameter in einem weiten Bereich variiert werden, ohne merkliche Einflüsse auf das Regelkreisverhalten.

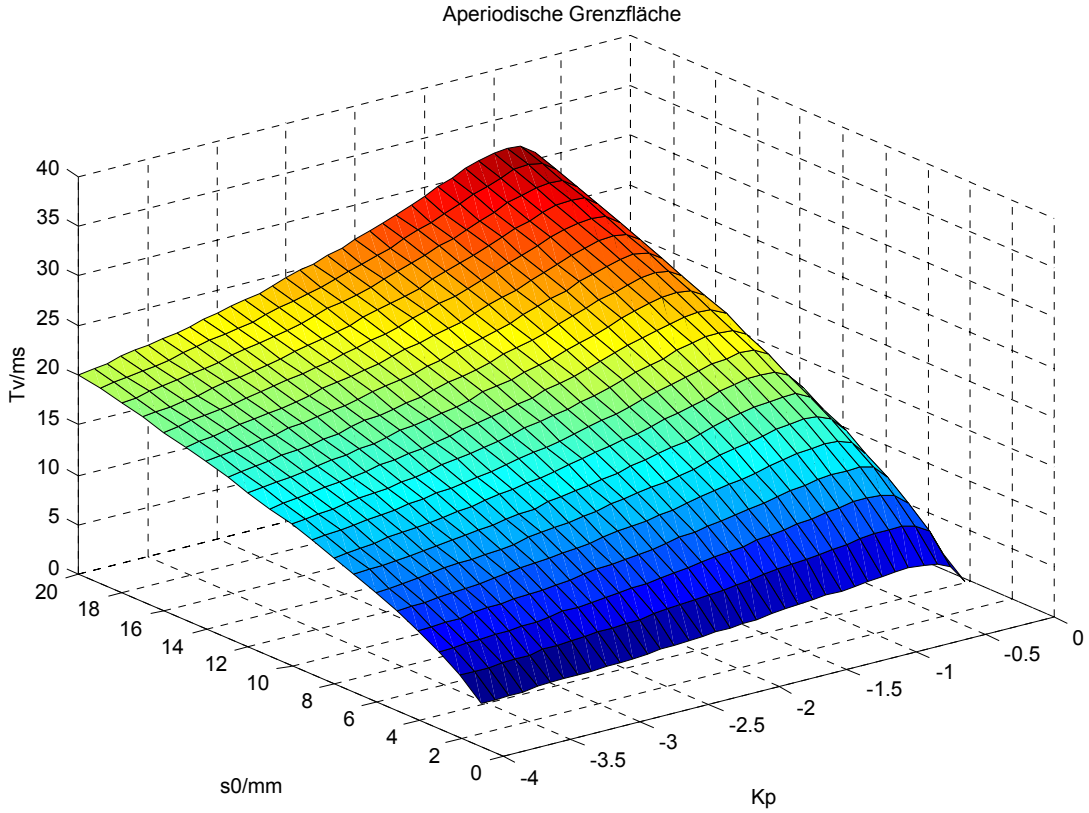
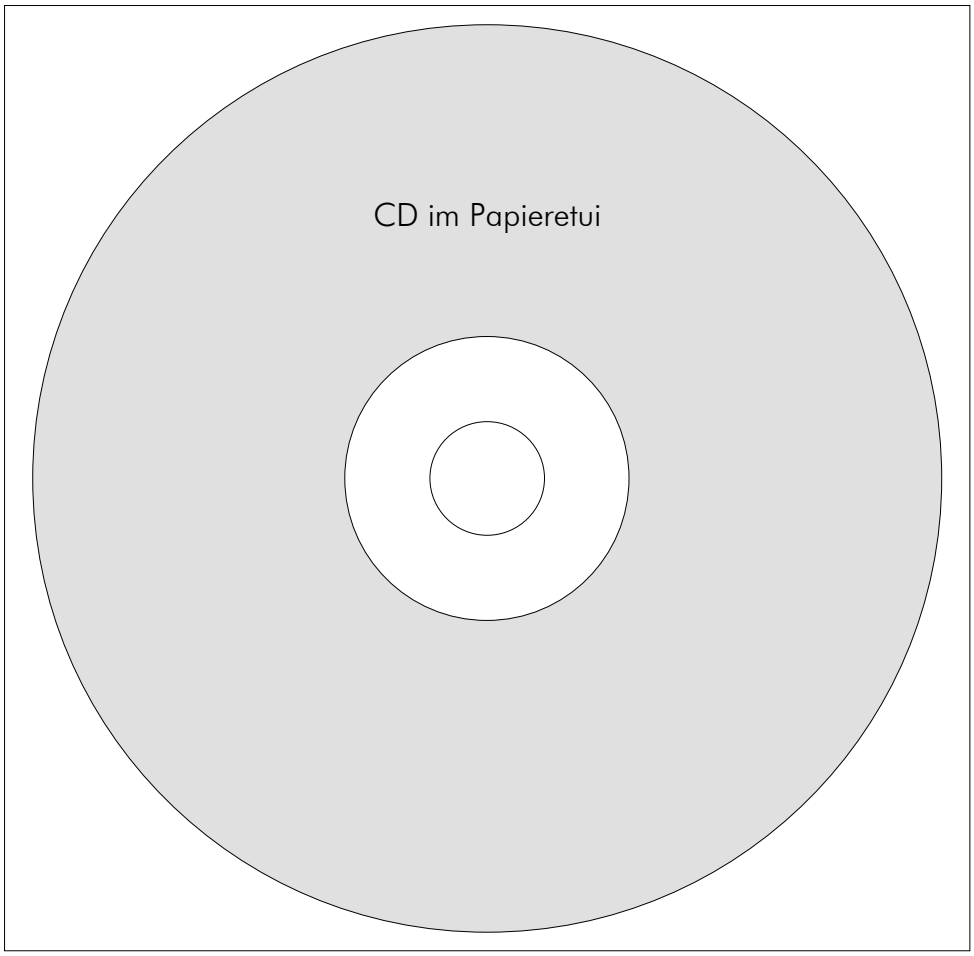


Bild 8.9: Aperiodische Grenzfläche für den PD-Regelkreis

8.13 Dokumenten- und Quellcode-CD

Auf dieser CD befinden sich die wichtigsten Dokumente, Quellcodes, Simulationsdateien, Bilder und Videos zu dieser Arbeit. Aus Portabilitätsgründen wurden die Dokumente entweder im PDF- oder „Raw"-Textformat gespeichert. In den PDF-Dateien befinden sich die verwendeten Schriften, so dass es zu keinen Darstellungsproblemen kommen sollte. Des Weiteren enthält die CD ein kurzes Video über das Demonetz im AVI-Format. Zahlreich Fotos über den Aufbau, die Knoten und die Internet-Anbindung im JPEG- oder GIF-Format ergänzen die Dokumentation. Der Autor übernimmt keine Garantie für die Fehlerfreiheit und Vollständigkeit der Quellcodes auf der CD und auch kein „Support". Außerdem sei darauf hingewiesen, dass sich durch die Weiterführung der Forschungsarbeiten Änderungen ergeben können.



9 Literatur

Bussysteme allgemein

[1] Anonym: Welcher Feldbus eignet sich für die Sensor-/Aktorebene? Informationsblatt der Anwendervereinigung „DIN-Meßbus“ e.V. der Universität Hannover in Zusammenarbeit mit der Deutschen Forschungsgesellschaft für die Anwendung der Mikroelektronik e.V.

[2] Anonym: INFIDA: Die Internet Fieldbus Datenbank. http://www.infoside.de/infida/inhalt_wissen.htm, BestWeb GmbH, 11.01.2001.

[3] Anonym: Introduction to In-Vehicle Networking. http://www.intel.com/design/auto/autolxbk.htm, Intel Corporation, 11.04.2001.

[4] Anonym: Siemens-Vorschlag zur IEC 61158. Fachzeitschrift atp, Heft 1/1999, S. 54.

[5] Böttcher, Jörg: EN 50170 - Die europäische Feldbusnorm / Teil 1. Fachzeitschrift Elektronik, Heft 12/1996, S. 70ff.

[6] Böttcher, Jörg: EN 50170 - Die europäische Feldbusnorm / Teil 2. Fachzeitschrift Elektronik, Heft 14/1996, S. 63ff.

[7] Böttcher, Jörg: EN 50170 - Die europäische Feldbusnorm / Teil 3. Fachzeitschrift Elektronik, Heft 16/1996, S. 48ff.

[8] Böttcher, Jörg: EN 50170 - Die europäische Feldbusnorm / Teil 4. Fachzeitschrift Elektronik, Heft 17/1996, S. 54ff.

[9] Dembowski, Klaus: Computerschnittstellen und Bussysteme. München: Markt-und-Technik-Verlag, 1993.

[10] Dönges, Eric: Me-Too: Architekturmodell eines verläßlichen Realzeitrechners für die Automatisierungstechnik. Diplomarbeit. Lehrstuhl für Realzeit-Computersysteme (ehemals Lehrstuhl für Prozessrechner), Technische Universität München, April 1999.

[11] Färber, Georg: Bussysteme: Parallele und serielle Bussysteme, lokale Netze. 2., überarb. u. erw. Aufl. München, Wien: Oldenbourg Verlag, 1987.

[12] Färber, Georg: Feldbus-Technik heute und morgen. Fachzeitschrift atp, Heft 11/1994, S. 16ff.

[13] Feicht, Matthias; Gillhuber, Andreas; Kink, Andreas; Naber, Martin; Klügl, Jürgen; Wech, Stefan; Zeller, Wolfgang: Feldbusse. Oberseminar Prozeßrechentechnik. Lehrstuhl für Realzeit-Computersysteme (ehemals Lehrstuhl für Prozessrechner), Technische Universität München, WS 1994/95.

[14] Huber, Robert; Pfaffeneder, Bernd: Gemischter Betrieb von RS485- und CAN-Transceivern an einem Bus. Fachzeitschrift Elektronik, Heft 13/2000, S. 66ff.

[15] Scherff, Birgit; Haese, Erwin; Wenzek, Hagen: Feldbussysteme in der Praxis: Ein Leitfaden für den Anwender. Berlin, Heidelberg, New York, Barcelona, Honkong, London, Mailand, Paris, Singapur, Tokio: Springer Verlag, 1999.

[16] Reißenweber, Bernd: Feldbussysteme. München, Wien: Oldenbourg Verlag, 1998.

TTP/A

[17] Anonym: Objective Interface Systems. TTTech Computertechnik AG and VERTEL Corporation, supported by Technische Universität Wien. Smart Transducers Interface OMG TC Document orbos/2001-06-03, July 2001.

[18] Eberle, Stefan: Internetfähige TTP/A-Werkzeuge. Vortrag auf der TTSB Projektabschlusspräsentation. Institut für Automatisierungs- und Softwaretechnik, Universität Stuttgart, Sept. 2001.

[19] Färber, G.; Huber; R.; Elmenreich, W.; Haidinger, W. u.a.: Smart Transducers „Kommunikationsarchitekturen für verlässliche Automatisierungssysteme". Förderantrag an die Bayerische Forschungsstiftung. Lehrstuhl für Realzeit-Computersysteme, Technische Universität München, Dez. 2001.

[20] Färber, G. u.a.: Kommunikationsarchitekturen für verlässliche Automatisierungssysteme (Time Triggered Sensor Bus TTSB). Förderantrag an die Bayerische Forschungsstiftung. Lehrstuhl für Realzeit-Computersysteme, Technische Universität München, Feb. 1998.

[21] Huber; Robert: Realzeit-Sensor/Aktorbus für „Embedded Systems". Fachkongress Embedded Intelligence, Feb. 2002.

[22] Kopetz, Hermann: Specification of the TTP/A Protocol. Confidential Draft, Version 1.4, Technical University of Vienna, May 2000.

[23] Pfaffeneder, Bernd: Time Triggered Sensor Bus. Vortrag auf der Abschlusspräsentation des Forschungsprojekts TTSB am Lehrstuhl für Realzeit-Computersysteme. SiemensVDO, Technische Universität München, Sept. 2001.

ASI, CAN

[24] Anonym: Actuator Sensor Interface (AS-Interface). Complete Specification Version 2.11. AS-International Association, Germany, 01.02.2000.

[25] Anonym: AS-Interface: International Web Site. http://www.as-interface.com, AS-International Association, 11.01.2001.

[26] Anonym: ASI: Busverbindung für Sensoren. http://www.infoside.de/infida/asi/asi000.htm, Leuze electronic, 23.01.2001.

[27] Anonym: CAN Specification Version 2.0: Part A and B. Bosch GmbH, 1991.

[28] Anonym: CiA Draft Standard 102 Version 2.0: CAN Physical Layer for Industrial Applications. CiA (CAN in Automation), April 1994.

[29] Anonym: ISO 11898-4: Road vehicles - Controller area network (CAN) - Part 4: Time-triggered communication. Preparatory Document. International Organization for Standardization (ISO), 1999.

[30] Anonym: CAN and CANKingdom Explained. CAN Kingdom International. The King's Pages, Volume 1, Issue 1, March 2001, Page 2.

[31] Anonym: CiA Homepage. http://www.can-cia.de, CAN in Automation (CiA), 10.05.2001.

[32] Anonym: CANpie Basic Information. http://www.microcontrol.net/CANpie/cp_basic.php3, MicroControl GmbH & Co. KG, 10.05.2001.

[33] Anonym: M3S Web Server. http://www.tno.nl/m3s, TNO Institute of Applied Physics, Netherlands, 16.05.2001.

[34] Anonym: CAN Data Link Layer. CiA (CAN in Automation), http://www.can-cia.de/candll.pdf, 10.01.2002.

[35] Etschberger, Konrad: CAN: Grundlagen, Protokolle, Bausteine, Anwendungen. München, Wien: Hanser-Verlag, 1994.

[36] Etschberger, Konrad; Schlegel, Christian: Der Schicht-7-Standard für CAN-Netzwerke: Beschreibung von Geräten durch CAL-basierende Profile. Fachzeitschrift Elektronik, Heft 7/1995, S. 78ff.

[37] Flaschka, Elmar: Binäre Sensoren am Bus. Fachzeitschrift Elektronik, Heft 12/1994, S. 64ff.

[38] Haimerl, Michael: Untersuchung zur Sicherheit und Zuverlässigkeit von Datenbussen im Kraftfahrzeug. Diplomarbeit. Fachhochschule Regensburg, Siemens AG, Juli 1998.

[39] Huber, Robert: Definition eines Netzwerkkonzeptes für die Finishgeräte der Firma Veit. Studie. Lehrstuhl für Realzeit-Computersysteme (ehemals Lehrstuhl für Prozessrechner), Technische Universität München, Juni 1996.

[40] Indefrey, Klaus; Stripf, Wolfgang: Sicherheit im Doppelpack. Fachzeitschrift Computer & Automation, Heft 1-2/2000, S. 50ff.

[41] Kastner, Wolfgang: Controller Area Network. Vorlesung Feldbussysteme. Institut für rechnergestützte Automation, Technische Universität Wien: www.auto.tuwien.ac.at/FBus/CAN.pdf, 11.01.2002.

[42] Kugelmann, Michael; Mayer, Michael; Zimmermann, Werner: Sprachübertragung über CAN: MSR-Daten und Sprache auf einer Leitung. Fachzeitschrift Elektronik, Heft 10/1999, S.114ff.

[43] Kriesel, Werner; Madelung, Otto W. (Hrsg.): ASI: Das Aktuator-Sensor-Interface für die Automation. München, Wien: Hanser-Verlag, 1994.

[44] Lawrenz, W.: CAN: Grundlagen und Praxis. Heidelberg: Hüthig Verlag, 1994.

[45] Reichel, Thomas: AS-Interface noch anwendungsfreundlicher. Fachzeitschrift Elektronik, Heft 5/2000, S. 60ff.

[46] Schmidt, Wolfgang; Speckmann, Herrmann: LBS in der Prüfung. http://www.dlg-frankfurt.de/thema3/lbs/lbs.htm, Deutsche Landwirtschaftsgesellschaft (DLG), 16.05.2001.

[47] Schofield, Mike: Controller Area Network (CANbus). http://www.omegas.co.uk/CAN, 16.05.2001.

[48] Unruh, Jan; Mathony, Hans-Jörg; Kaiser, Karl-Heinz: Error Detection Analysis of Automotive Communication Protocols. SAE International Congress, 1990.

[49] Wenkebach, Ullrich: CAN in der Medizintechnik: Sensor-/Aktor-Bus auf der Basis von CAN. Fachzeitschrift Elektronik, Heft 16/1997, S. 34ff.

[50] Zeltwanger, Holger: Wir hatten andere Märkte im Visier: Wie vor 15 Jahren der Siegeszug des seriellen Bussystems CAN begann. Fachzeitschrift Elektronik, Heft 3/2001, S. 58ff.

[51] Zeltwanger, Holger: CAN - ein Bussystem nicht nur für Kraftfahrzeuge. http://www. micro-control.net/de/can/can_intro_text.php3, MicroControl GmbH & Co. KG, 10.05.2001.

LIN, PPC, ISO 9141, DC-Bus

[52] Anonym: LIN Protocol Specification. Revision 1.2. Motorola GmbH, Germany, 17. Nov. 2000.

[53] Anonym: LIN - Local Interconnect Network. http://www.lin-subbus.org, LIN - Homepage, 18.10.2001.

[54] Anonym: DC-Bus for Battery Power Line Commuincation. Yamar Electronics. http://www.yamar.com/Default.htm, 08.11.2001.

[55] Anonym: DC-Bus for Battery Power Line Commuincation. The Israel Automotive, Subcontracting, Metal Newsletter. http://www.export.org.il/newsletter/metal/metalonline4.html, 8.11.2001.

[56] Anonym: QDC10 – Multiplexed Data On Battery Power Lines. Preliminary Informations (Data Sheet), Rev. 1.1. Yamar Electronics.

[57] Anonym: DCB250 – High Speed Battery Power Lines Communication. Preliminary Informations (Data Sheet), Rev. 1.03. Yamar Electronics.

[58] Anonym: DIN ISO 9141: Straßenfahrzeuge Diagnosesysteme. Anforderungen für den Austausch digitaler Informationen. Deutsche Institut für Normung e.V. (DIN), April 1992.

[59] Anonym: All your Networks Simultaneously. http://www.intrepidcs.com/neovi/net_over. htm, Intrepid Control Systems Inc., 12.04.2001.

[60] Kettig, G.; Beer, S.; Kugler, A.; Kolb, J.; Brandt, T.: Dezentrale Bussysteme im Kfz: Serielle Kfz-Busse ohne teuren Overhead. ZVEI-Podium electronica 2001, Nov. 2000.

[61] Lauterbach, K. D.: Busse in Serie: Kosten sparen mit seriellen Bussystemen im Automobil. Fachzeitschrift F&M Feinwerktechnik, Mikrotechnik, Mikroelektronik. Heft 5/2001.

[62] Mayer, Bettina: Ein Bus fürs Auto. Zeitschrift Focus, Heft 50/2000, S. 205f.

[63] Specks, J. W.; Rajnák, A.: LIN - Protocol, Development Tools, and Software Interfaces for Local Interconnect Networks in Vehicles. 9th International Conference on Electronic Systems for Vehicles, Baden-Baden, Oct. 2001.

BST, DSI, Planet, SafetyBus (Delphi)

[64] Anonym: DSI Bus Standard. Release 1.0. Motorola, TRW, 1999.

[65] Anonym: A short introduction in DSI (Distributed Systems Interface). http://www.mot-sps.com/automotive, 27.05.1999.

[66] Anonym: In-Vehicle Networking. Philips Semiconductors. http://www-us.semiconductors. philips.com/automotive/ivn, 08.11.2001.

[67] Bogenrieder, H.; Bühring, H.; Pfaffeneder, B.: Common Bosch Siemens Temic BUS DESCRIPTION. Rev. 2.00. BST-Spezifikation: Bosch, Siemens, Temic, April 2001. www.temic.de/e/prod/insass/Bosch_Siemens_Temic_200.pdf, 03.01.2002.

[68] Bühring, Peter; Jöhnk, Egon: Planet 2.1: Preliminary System Specification. Technical Report. Philips Semiconductors, April 2000.

[69] Huber, Robert; Färber, Georg: Vorstudie zur Sicherheitsanalyse des gemeinsamen Zündbusses von Bosch, Siemens und Temic (BST-Bus). Vorstudie. Lehrstuhl für Realzeit-Computersysteme, Technische Universität München, August 2000.

[70] Jost, Kevin: First dedicated safety bus from TRW. SAE International, http://www.sae.org/ automag/convergence/trw.htm, 08.11.2001.

[71] Lupini, Christopher:Multiplex Bus Progression. SAE Technical Paper Series. SAE International. http://www.delphiauto.com/pdf/techpapers/2001-01-0060.pdf, 08.11.2001.

MOST, D2B, IDB

[72] Anonym: MOST: Media Oriented Systems Transport. Specification, Rev 2.0. Most Cooperation, 12.1999.

[73] Anonym: IDB Background. http://www.sae.org/technicalcommittees/back.htm. SAE International, 05.11.2001.

[74] Anonym: SAE J2366 (Draft): ITS Data Bus (IDB) Protocol. Society of Automotive Engineers (SAE), Dec. 2000. http://its-standards.net/Documents/J2366r2.pdf, 06.11.2001.

[75] Anonym: Fachwissen allgemein: D2B. http://www.innovative-systems.de/dglossar.htm, 06.11.2001

[76] Panzer, Klaus; Müller, Gustav; Hurt, Hans; Thiel, Christian: A Multimedia Infrastructure for Automotive Applications from D2B to MOST. http://www.spacetools.com/site/contact_v3_8/ V3_8p49.pdf, Infineon Technologies, 05.11.2001.

[77] Spelt, Philip; Kirson Allan: IST Data Bus Demo98-99. 17th Annual DASC, November 4, 1998. http://www.epm.ornl.gov/~sfp/DASC_IDB.pdf, 05.11.2001.

[78] Tappe, Robert; Thiel, Christian; König, Rainer: MOST - Media Oriented Systems Transport: Standard für Multimedia im Automobil etabliert. Fachzeitschrift Elektronik, Heft 14/2000, S. 54ff.

Byteflight, FlexRay, TTP/C

[79] Anonym: Byteflight Homepage. http://www.byteflight.com/homepage.htm, 05.11.2001.

[80] Anonym: FlexRay Homepage. http://www.flexray-group.com, FlexRay Group, 05.11.2001.

[81] Anonym: TTTech Homepage. TTTech AG. http://www.tttech.com, 05.11.2001.

[82] Anonym: TTP/C Homepage. Institut für Technische Informatik, Technische Universität Wien. http://www.vmars.tuwien.ac.at/projects/ttp/ttpc.html, 06.11.2001.

[83] Anonym: TTPforum. TTTech AG. http://www.ttpforum.org, 07.11.2001.

[84] Anonym: Introducing TTP. TTP News. TTTech AG (ehemals GmbH), 1999.

[85] Anonym: Comparison: CAN vs. Byteflight vs. TTP/C. TTTech AG (ehemals GmbH), April 2000.

[86] Anonym: Drive-by-Wire-Technologie wird kommen. Fachzeitschrift Elektronik, Heft 24/1998, S. 27ff.

[87] Gagea, Leonard: Austria Mikro Systeme and TTTech develop next generation TTP/C-C2 communication controller – AS8202. TTTech AG. http://www.tttech.com, 05.11.2001.

[88] Grießbach, Robert; Berwanger, Josef; Peller, Martin: byteflight – neues Hochleistungs-Datenbussystem für sicherheitsrelevante Anwendungen. Sonderausgabe der Fachzeitschrift Automotive Electronics der ATZ/MTZ, Januar 2000.

[89] Higgelke, Ralf: Kommunikationsprotokoll für bis zu 10 MBit/s: Der Manta ist zurück. Fachzeitschrift Design & Elektronik, Heft 02/2001, S. 92.

[90] Kopetz, H.: A Comparison of TTP/C and FlexRay. Research Report. Institut für Technische Informatik, Technische Universität Wien, May 2001.

[91] Poledna, Stefan; Kroiss, Georg: Zeitsteuerung ersetzt Ereignissteuerung: TTP läutet den Paradigmenwechsel der Echtzeitkommunikation ein. Fachzeitschrift elektronik industrie, Heft 12/1998, S. 82ff.

[92] Poledna, Stefan; Kroiss, Georg: TTP: „Drive by Wire“ in greifbarer Nähe. Fachzeitschrift Elektronik. Heft 14/1999, S. 36ff.

[93] Poledna, Stefan; Stöger, Georg; Schlatterbeck, Ralf; Niedersüß, Michael: Sicherheit auf vier Rädern. Time-Triggered Protocol – Serieneinsatz in Flugzeug und Automobil. Fachzeitschrift Elektronik, Heft 10/2001, S. 114ff.

Weitere Busse

[94] Anonym: JetWeb Einführung: Warum eine neue Steuerungstechnologie? http://www.jetweb-technology.com, Jetter AG, 05.11.2001.

[95] Anonym: Electric Railway Equipment - Train Bus - Part 1: Train Communication Network. IEC 61375-1. International Electrotechnical Commission (IEC), 1999.

[96] Claessen, Ulrich; Kirrmann, Hubert: Feldbus für Europas Züge: Sichere Datenübertragung mit dem Train Communication Network. Fachzeitschrift Elektronik, Heft 17/2000, S. 40 ff.

[97] Gotzhein, R.: Forschungsergebnisse: Maßschneiderung von Kommunikationssystemen. Computer Networks Group, Fachbereichs Informatik, Universität Karlsruhe, März 2001.

[98] Rushby, John: A Comparison of Bus Architectures for Safety-Critical Embedded Systems. Technical Report. Computer Science Laboratory, SRI International: Sept. 2001.

Codierung, Sicherheit

[99] Anonym: BIA-Homepage. http://www.hvbg.de/d/bia/pub/rep/bia0697.htm. Berufsgenossenschaftliches Institut für Arbeitssicherheit, 21.05.2001.

[100] Conrads, Dieter: Datenkommunikation: Verfahren, Netze, Dienste. 3. überarb. u. erw. Aufl. Braunschweig, Wiesbaden: Vieweg & Sohn Verlagsgesellschaft mbH, 1996.

[101] Haidinger, Wolfgang; Huber Robert: Generation and Analysis of the Codes for TTP/A Fireworks Bytes. Technical Report. Institut für Technische Informatik, Technical University Vienna Austria; Institut for Real-Time Computersystems, Technical University Munich Germany, May 2000.

[102] Hänsler, Eberhard: Grundlagen der Theorie statistischer Signale. Berlin, Heidelberg, New York: Springer Verlag, 1983.

[103] Montenegro, Sergio: Sichere und fehlertolerante Steuerungen: Entwicklung sicherheitsrelevanter Steuerungen. München, Wien: Hanser Verlag, 1999.

[104] Schneider-Obermann, Herbert: Kanalcodierung: Theorie und Praxis fehlerkorrigierender Codes. Braunschweig, Wiesbaden: Vieweg & Sohn Verlagsgesellschaft mbH, 1998.

[105] Scheurer, Christian: Fehlererkennung: Cyclic Redundancy Check Code, Odd/Even Parity, Checksum. Technischer Bericht. http://www.mountpoint.ch/~unique/project/crc/index. html, 28.03.2001.

[106] Söder, Günther: Modellierung, Simulation und Optimierung von Nachrichtensystemen. Berlin, Heidelberg, New York, London, Paris, Tokyo, Hong Kong, Barcelona, Budapest: Springer, 1993.

[107] Schumny, Harald: Signalübertragung. Lehrbuch der Nachrichtentechnik mit Datenfernverarbeitung. Braunschweig, Wiesbaden: Vieweg Verlag, 1987.

[108] Swoboda, Joachim: Codierung zur Fehlerkorrektur und Fehlererkennung.München, Wien: Oldenbourg Verlag, 1973.

[109] Wicker, Stephen: Error Control Systems for Digital Communication and Storage. Englewood Cliffs, New Jersey: Prentice-Hall Inc., 1995.

[110] Williams, Ross: The CRC Pitstop. http://www.ross.net/crc, 28.03.2001.

Bausteine

[111] Anonym: micoModul-166: Hardware-Manual. Phytec AG, Januar 1999.

[112] Anonym: micoModul-165: Hardware-Manual. Phytec AG, Januar 1999.

[113] Anonym: C165/C163: 16-Bit CMOS Single-Chip-Microcontrollers. User's Manual 10.96, Version 2.0. Siemens AG, Okt. 1996.

[114] Anonym: Microcomputer Components: SAB 80C166/83C166: 16-Bit CMOS Single-Chip-Microcontrollers for Embedded Control Applications. User's Manual, Revision 6.90. Siemens AG, Juni 1990.

[115] Anonym: C166S Product Overview. Infineon AG.

[116] Anonym: Infineon stellt neuen 16-Bit-Mikrocontroller-Core vor. http://www.infineon.com/news/press/011_012d.htm, Infineon AG, 14.07.2001.

[117] Anonym: ST10V168: 16-Bit MCU with 256K Byte FLASH Memory and 8K Byte RAM. Datasheet. STMicroelectronics, August 2000.

[118] Anonym: ST62T18C/E18C: 8-Bit MCUs with A/D Converter, Auto-Reload Timer, UART, OSG, Safe Reset and 20-PIN Package. Datasheet, Revision 2.5. STMicroelectronics, Nov. 1999.

[119] Anonym: 8-bit AVR Microcontrollers with 2K Bytes of In-System Programmable Flash: AT90S2313. Datasheet, Revision 0839F-10/00. Atmel Corporation, Oct. 2000.

[120] Anonym: AT91 ARM Thumb Microcontrollers: AT91M40400. Datasheet, Revision C. Atmel Corporation, Feb. 1999.

[121] Anonym: PIC12C5xx: 8-Pin, 8-Bit CMOS Mircocontrollers. Datasheet. Microchip Inc., 1999.

[122] Anonym: PIC16F62x: FLASH-Based 8-Bit CMOS Mircocontrollers. Datasheet. Microchip Inc., 1999.

[123] Anonym: PIC16C745/765: 8-Bit CMOS Mircocontrollers with USB. Preliminary Datasheet. Microchip Inc., 2000.

[124] Anonym: PIC16F8X: 8-Bit CMOS FLASH/EEPROM Mircocontrollers. Datasheet. Microchip Inc., 1996.

[125] Anonym: 87LPC767: Low power, low price, low pin count (20 pin) microcontroller with 4 kB OTP and 8-bit A/D. Datasheet. Philips Semicondutors, Feb. 2000.

[126] Anonym: HC05: MC68HC05B6 (HCMOS) Microcomputer UNIT. Technical Data, Rev. 4. Motorola Ltd., 1999.

[127] Anonym: μPD78F9076: 8-Bit Single-Chip Microcontrollers. Preliminary Product Information. NEC Corporation, June 2000.

[128] Anonym: Z86C04/08: CMOS 8-Bit Low-Cost 1K/2K-ROM Microcontrollers. Preliminary Product Specification. Zilog, 1997.

[129] Anonym: Lieferübersicht DSL: Quarz- und Oszillator Produkte. DSL GmbH, 2000.

[130] Anonym: Ceramic Resonators. http://www.rxdtech.com/resonators.html, RXD Technologies, 17.09.2001.

[131] Anonym: Ceramic Resonators: Characteristics. http://www.spkecl.com/htdoc/ceramic6bc.htm, SPK Electronics, 17.09.2001.

[132] Born, Gabriele: 16-bit-Controller liefert 23-Power. Fachzeitschrift Elektronik, Heft 3/2001, S. 52ff.

[133] Wagner, Reinhardt: Design von UART-Quarzoszillatoren. Fachzeitschrift Elektronik, Heft 22/2000, S. 168ff.

Sonstiges

[134] Anonym: On-Line C166/ST10 Technical Documents: Auto Baudrate Detection On The C16x. http://www.hitex.co.uk/c166/autobuad.html. Hitex UK, Oct. 2001.

[135] Böhmer, Erwin: Elemente der angewandten Elektronik. 10., überarb. Aufl. Braunschweig; Wiesbaden: Vieweg Verlag, 1996.

[136] Färber, Georg: Realzeitsysteme. Manuskript zur Vorlesung. Lehrstuhl für Realzeit-Computersysteme, Technische Universität München, WS 2001/2002.

[137] Föllinger, Otto: Regelungstechnik: Einführung in die Methoden und ihre Anwendung. 6., vollst. überarb. Aufl. Heidelberg: Hüthig Verlag, 1990.

[138] Göhner, P.: Komponentenbasierte Entwicklung von Automatisierungssystemen. VDI Berichte 1397: GMA-Kongress ´98 Mess- und Automatisierungstechnik. VDI Verlag, 1998, S. 513ff.

[139] Goodhue, Greg: Automatic baudrate detection for the 80C51. Application Note AN447. Philips Semiconductors, June 1993.

[140] Huber, Robert; Burmberger, Gregor: Regelung eines schwebenden Körpers unter Windows NT. Fachzeitschrift Elektronik, Heft 18/2000, S. 90ff.

[141] Huber, Robert; Münnich Alexander:Praktikum Realzeitprogrammierung. Skript. Lehrstuhl für Realzeit-Computersysteme, Technische Universität München, SS 2000.

[142] Möschwitzer, Albrecht et. al.: Formeln der Elektrotechnik und Elektronik / Netz. München, Wien: Hanser Verlag, 1986.

[143] Oertel, Klaus: Markttreiber Kfz-Elektronik. Fachzeitschrift elektronik industrie, Heft 7/1999, S. 3.

[144] Schmidt, Günther: Grundlagen der Regelungstechnik: Analyse u. Entwurf linearer u. einfacher nichtlinearer Regelungen sowie diskreter Steuerungen. 2., überarb. u. erw. Aufl. Berlin, Heidelberg, New York, London, Paris, Tokyo: Springer Verlag, 1987.

[145] Tietze, Ulrich; Schenk, Christoph: Halbleiter-Schaltungstechnik. 9., neu bearb. u. erw. Aufl. Berlin, Heidelberg, New York, London, Paris, Tokyo, Hong Kong, Barcelona: Springer, 1990.

[146] Wenzel, Tobias: CAN Baudrate Detection with Infineon CAN devices. Microcontrollers Application Note AP2925, Infineon, July 1999.

[147] Willms, André: C-Programmierung: Programmiersprache, Programmiertechnik, Datenorganisation. Bonn: Addison-Wesley-Longman Verlag, 1996.