Hauptseminar zum Thema

Funktion des Automobilkommunkationssystems

Flexray

Stefan Aschenbach

23. Oktober 2006

Abbildung 1: Flexray Logo

Hauptseminar zum Thema
Funktion des Automobilkommunkationssystems
Flexray

Inhaltsverzeichnis

1 Vorwort

Diese Ausarbeitung beschääftigt sich mit dem als neuen Standard füür die Automobilkommunikation vorgesehene Technik *Flexray* . Sie basiert hauptsäächlich auf die offiziellen Spezifikationen, da diese als einzig güültige Referenz angenommen werden köönnen. Leider existieren noch keine gröößeren Publikationen (im Rahmen der Ausleihmöglichkeiten unserer Uni-Bibliothek) zu diesem Thema. Im Internet zu findende Themen sind meiSSt recht deutlich an die Eigenbeschreibung der *Flexray* Corporation angelehnt und somit kein Quell zusätzlicher Informationen.
Das Ziel dieser Ausarbeitung ist eine Zusammenfassung der angewandten Techniken in einem MaSSe, wie es zum Eingliedern in eine Vorlesungsveranstaltung zum Beispiel als Ergänzung oder Ersatz zum bestehen CAN Bus verwendet werden kann. Hierzu soll diese Ausarbeitung als Richtlinie dienen, sowie die beigefügten Folien zur Präsentation und als druckbares Skript.
Diese Ausarbeitung enthält insbesondere viele Vereinfachungen und verzichtet bewusst auf Details, die nur zur wirklichen Implementation, aber nicht zum Verständnis nötig sind. Es wurde versucht, Aufbau und Inhalt an die bestehenden Rechnernetze-Vorlesungen anzupassen.
Für explizite Zustands- und Variablennamen sowie Werte von Konstanten verweise ich hiermit auf die Spezifikationen (siehe Quellenverzeichnis, Seite 24).

2 Überblick

2.1 Motivation

Erst seit kurzer Zeit spielen elektronische Signale im Automobilbereich eine gröSSere Rolle. Anfangs wurde Elektronik praktisch ausschlieSSlich als Interface zum Fahrer gebraucht, zum Beispiel: digitale Anzeigegeräte, Warn- und Hinweissignale, Bedienung von Multimediageräten. Für diese Anwendung war kein ausgereiftes Bussystem nötig, dedizierte Steuerleitungen reichten völlig aus.
Mit der Zunahme an Sensorik und aktiven Fahrhilfen wie ABS und ESP entwickelte sich das CAN-Bus Konzept zum Quasi-Standard. Als Bussystem hatte eine zentrale Recheneinheit Zugriff auf die verteilten Sensor- und Aktorelemente der Fahrwerks- und Motortechnik.
Heute stöSSt auch diese Technik langsam an ihr Grenzen. Sensoren liefern immer feinere Daten, die mehr Bandbreite benötigen. Aktoren greifen immer genauer in sehr sensible Bereiche der Fahrzeugsteuerung ein und benötigen kurze Latenzzeiten und eine definierte Ausfallsicherheit und Fehlertoleranz. Auch kommt man vom zentralen Steuerrechner ab und verteilt dessen Aufgaben an dedizierte Einzelprozessoren, zusätzlich werden Multimediasysteme mit eingebunden, um Fahrzeugdaten zum Fahrer zu übermitteln. Hauptschlagworte in dieser Entwicklung ist die drive-by-wire Technik.
Die neuen Anforderungen an ein Automobil-Bussystem sind zusätzlich eine schnelle (Weiter-)Entwicklung mit EinfluSSnahme der Automobilhersteller, insbesondere die Möglichkeit der modularen Erweiterbarkeit und eines offenen Interfaces. Unter wirtschaftlichen Aspekten sollte ein neues System nicht-exclusiv allen zur Verfügung stehen und keine weiteren Lizensgebühren anfallen.

2.2 Zielsetzung der Flexray Entwicklung

Flexray ist ein neuartiges Protokoll zur Signalübermittlung im automobilen Bereich. Das Grundsystem ist unabhängig von der unterliegenden physikalischen Schicht, hat allerdings gewisse Anforderungen an diese. Als physikalisch Ebene für den ersten Testeinsatz wird zunächst eine 10MBit/s Shielded Twisted Pair Verdrahtung vorgesehen.
Das System unterstützt unterschiedlichste Topologiearten, wie zB Stern, Bus und vielfältig gemischte Verdrahtungsarten. Zur Verdrahtung können 2 parallele, physikalische Kanäle zur Steigerung der Übertragungsbandbreite oder zur Redundanzgenerierung benutzt werden.
Die Übertragungsrate liegt bei mindestens 10 Mbit/s wird vom Protokoll nach oben hin aber nicht begrenzt. Es liegt aber natürlich eine Abhängigkeit vom physikalischen Medium vor.
Das Protokoll unterstützt hohe Anforderungen im Bereich der Fehlertoleranz und -vermeidung, wie sie von sensiblen Echtzeitanwendungen im Automonilbereich benötigt werden. Signale werden redundant übertragen und unterliegen einer ständigen Synchronisation, ein kurzzeitiger Ausfall diese Mechanismen wird aber trotzdem verkraftet.
Die Überwachung der Übertragungen auf dem Bus kann sowohl zentral als auch dezentral in den einzelnen Geräten erfolgen.

Weitere Eigenschaften des Systems sollten sein:

- fehlertolerante Takt-Synchronisierung
- kollisionsfreier Bus-Zugriff
- garantierte Latenzzeiten
- Fehlertolleranz skalierbar (Single- oder Dualchannel System)
- zentral steuerbare Energieversorgen (Sleep und StandBy Modus)

2.3 Mitglieder der Foundation

Gründungsmitglieder der *Flexray Foundation* waren

- BMW, DaimlerChrysler, General Motors *als Anwender*
- Phillips und Motorola[1] *als Hersteller*
- Bosch *als Systementwickler*

Als weitere *Core Members* stieSSen später Freescale und Volkswagen hinzu.

Weitere Teilnehmer, sogenannte *Premium Associate Members* mit freier Nutzung des Systems und EinfluSS auf die Entwicklung für etwa 15.000 EURO im Jahr sind
Ford, Mazda, Fiat, Honda, Hyundai, Nissan, Peugeot-Citroen, Renault, Toyota sowie 5 weitere Firmen

Sowie ca[2]. 65 weitere *Associate Members* mit lediglich freier Nutzung für 7.500 EURO im Jahr.

Desweiteren gehören der Foundation noch ca. 43 *Development Members* an, darunter dSpace, MicroSys und 3Soft.

[1] z.Z. nur Premium Associate Member
[2] Stand Dez. 2005

3 Aufbau

3.1 Übersicht

Der allgemeine Aufbau eines Knotens im *Flexray* Netzwerk stellt sich wie folgt dar:

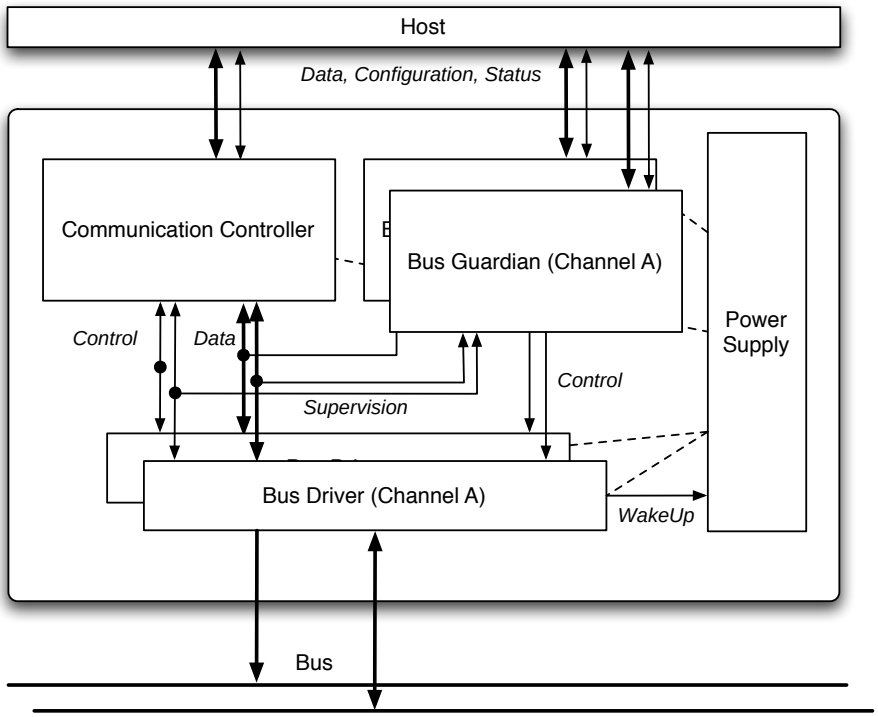

Abbildung 2: Aufbau eines Knotens

Hauptaugenmerk liegt hier auf dem *Communication Controller*, er steuert die Datenübertragung nach dem *Flexray*protokoll zwischen Host und Datenbus. Er ist auch für die Überwachung der lokalen Taktungen zur Synchronisation verantwortlich.
Der *Bus Guardian* überwacht die vom *Communication Controller* versendeten Daten und verhindert im Fehlerfall die Übertragung zum Bus durch blockieren des *Bus Driver*. Zur Überprüfung der Richtigkeit erhält er die selben Daten wie der *Communication Controller* vom *Host* und führt zusätzlich eine eigene Zeitsynchronisation durch.
Der *Bus Driver* übermittelt die Daten dann passend zum jeweiligen physikalischen Medium. Zusätzlich erkennt er bestimmte Bus-Signale und leitet sie als Steuerinformationen zur *Power Supply* und dem *Communication Controller* weiter, um auf Sleep oder WakeUp Befehle zu reagieren. Es existiert je ein *Bus Driver* für jeden physikalischen Kanal.

3.2 Netzwerktopologien

Flexray unterstützt mehrere Topologiearten auf physikalisch/logischer Ebene. Alle Angaben zu Anzahl an Knoten und Kabellängen beziehen sich auf den zur Zeit[3] angepeilten Standard von 10Mbit/s pro Kanal über Twisted Pair Verdrahtung.

1. **Basistypen**

   (a) Passiver Bus

   - Maximal 8 Knoten
   - Maximal 12 Meter Kabellänge zwischen den Knoten
   - Nicht fehlertolerant: Fällt der Bus aus, ist kein Knoten mehr erreichbar

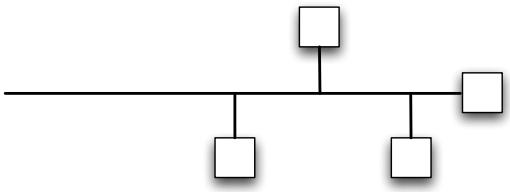

Abbildung 3: Passiver Bus

   (b) Stern

   - Maximal 16 Knoten an einem Stern
   - Maximal 12 Meter Länge eines Zweiges
   - Autonomes Routing und Power Management möglich
   - fehlertolerant: Ausfall nur eines Knotens bei Schaden am Bus

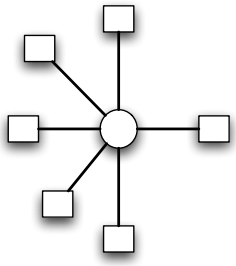

Abbildung 4: Stern - Bus

[3] Stand:29.12. 2005

2. **Mischformen**

   (a) Kaskadierter Stern

   - Maximal 3 kaskadierte Sterne
   - Die unterschiedlichen Nachrichtenverzögerungen werden durch die Synchronisation ausgeglichen

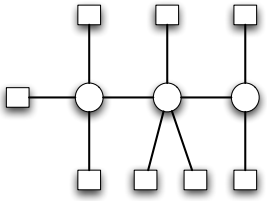

Abbildung 5: Kaskadierter Stern - Bus

   (b) Stern mit passivem Bus

   - Ein passiver Bus kann einen beliebigen einzelnen Knoten ersetzen

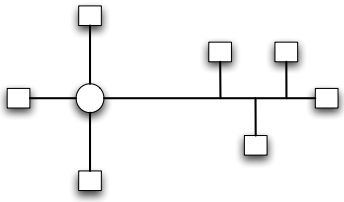

Abbildung 6: Stern mit passivem Bus

3. **Zwei-Kanal Formen**

   - Beliebige Mischung aus Einkanal- und Mehrkanalknoten
   - Beliebige Mischung der Topologien
   - Nutzung als redundante Verbindung zu Erhöhung der Fehlertoleranz oder zur Verdopplung der Bandbreite

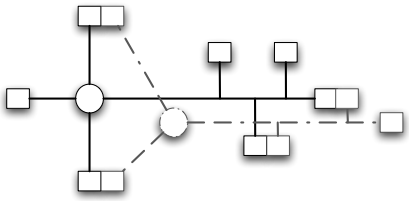

Abbildung 7: Zwei-Kanal Beispiel

3.3 Bus Driver

Die elektrische Signalübertragung basiert auf drei Zuständen. Es werden zwei Leitungen zur Generierung dieser Zustände benutzt. Die Spannungslagen sind dabei symmetrisch um 2,5V.

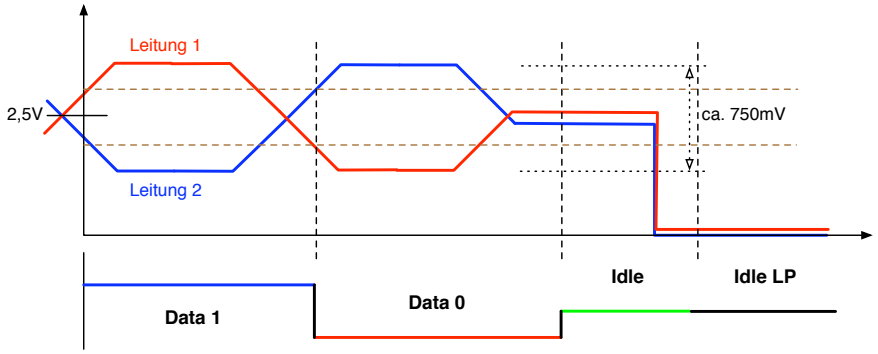

Abbildung 8: Erzeugung der log. Zustände

Sowohl Data 0 als auch Data 1 sind dominant, setzen sich also gegenüber Idle durch. Das Erkennungslevel im empfangenden Knoten ist wesentlich geringer als die Aussteuerung der Signale (bessere Erkennung auch über lange Signallaufzeiten). Beim Erkennen des Idle-Zustandes reagiert der Empfänger mit einer Verzögerung, was allerdings kleinere Störungen auf der Leitung bereits ausgleicht beziehungsweise ignoriert. Der Idle LP (Low Power) Zustand ist ein Fehlerzustand, wenn keine Spannung am Bus anliegt, beziehungsweise der Output, den ein *Bus Driver* ohne Spannungsversorgung liefert.
Der Bus Driver regelt zudem die Power Modes des Knotens. Mittels dedizierter Steuerleitung direkt zur *Power Supply* oder über ein Symbol auf der Datenleitungen zum *Communication Controller* steuert er die drei Zustände des Netzwerkknotens:

- *Normal:* normaler Betrieb aller Module sowie des Hosts
- *Standby:* Host arbeitet weiter, Kommunikationsmodule sind abgeschaltet
- *Sleep:* Alle Module inkl. Host sind abgeschaltet

Das WakeUp Symbol auf der Datenleitung, die auch im Sleep-Modus noch überwacht wird, besteht aus einem mind. 6 μs langen Data 0, einer mind. 6 μs langen Idle-Pause und einem weiteren 6 μs Data 0. Das ganze in einem Zeitslot von weniger als 30 μs. Siehe dazu auch ”Symbolübertragung”, Kapitel 4.2.3, Seite 18.

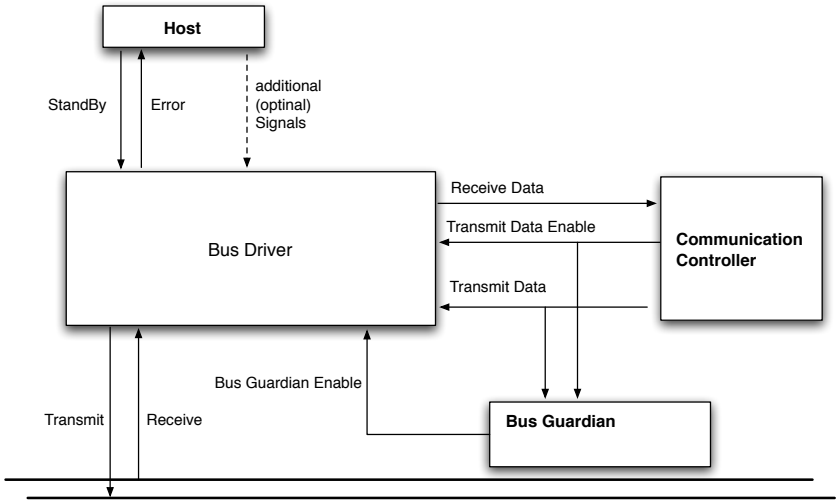

Abbildung 9: Kommunikation des Bus Driver

3.4 Communication Controller

Der *Communication Controller* implementiert die eigentlichen Protokollaktionen. Er nimmt vom *Host* Informationen entgegen und verpackt sie als *Flexray* kompatible Nachrichten. Umgekehrt werden ankommende Nachrichten dekodiert und dem *Host* übergeben. Zusätzlich nimmt der Controller Konfigurationsdaten vom *Host* entgegen und gibt Statusinformationen zurück. Dieser Austausch erfolgt über das *Controller Host Interface (CHI)*

Zu den Haupaufgaben des *Communication Controller* zählen:

- Enkodierung/Dekodierung
- Zugriffssteuerung auf das Medium
- Symbolerkennung und -verarbeitung
- Zeitsynchronisation

Insbesondere stellt der *Communication Controller* sicher, daSS alle vom *Host* ausgelösten Aktionen und beim Wiederaufnehmen der Kommunikation nach Sleep-Modi korrekt zu den normalen Busaktivitäten synchronisiert werden.

Die Überwachung des *Communication Controller* durch den *Bus Guardian* erfolgt nicht über eine zusätzliche Schnittstelle, sondern durch Abhören der Datenleitung zum *Bus Driver*. Sperrt der *Bus Guardian* die Übertragung nach einem Fehler, wird dies aber dem *Communication Controller* übermittelt, der dann eine Statusmeldung an den *Host* weitergibt und auf den Fehler reagieren kann.

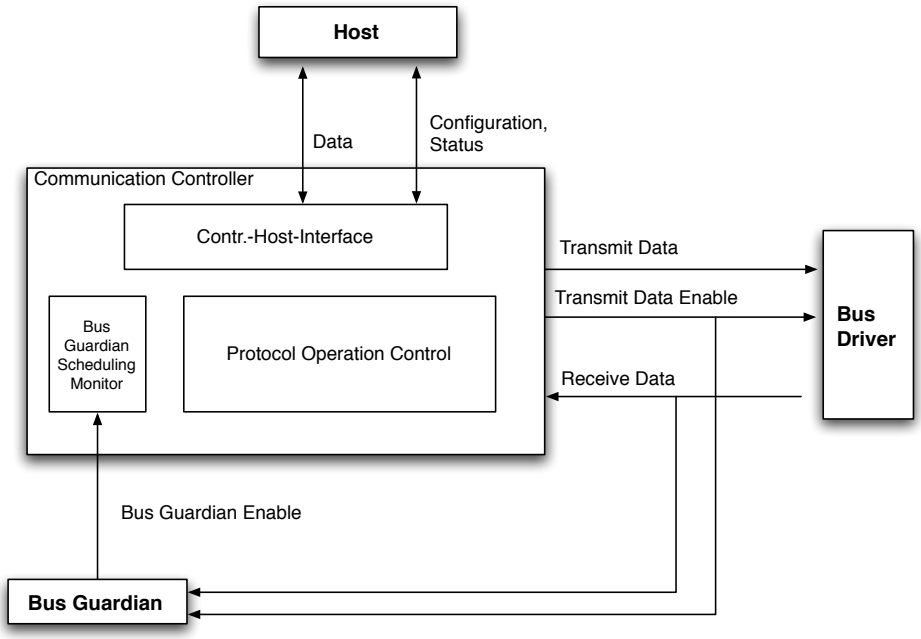

Abbildung 10: Aufbau und Kommunikation des Communication Controllers

3.5 Bus Guardian

Die Spezifikation des Bus Guardian *ist zur Zeit noch nicht vollständig abgeschlossen. Es wird versucht im Nachhinein die vorgebenden Anforderungen mit möglichst wenigen Änderungen an bisherigen Spezifikationen umzusetzen. Leider gelingt das nicht immer. Es könnten also durchaus kleine Unstimmigkeiten zu oben beschriebenen Funktionsweisen auftreten, die aber leider noch nicht genauer gelöst sind.*

Ein *Bus Guardian* ist generell optional. Er kann als zentrales Gerät am Bus hängen (Central Bus Guardian), oder in die einzelnen Knoten integriert sein (Local Bus Guardian). Natürlich lassen sich diese Konfigurationen auch kombinieren und sogar mehrere CBGs können im selben Netz eingesetzt werden. Ein *Bus Guardian* arbeitet dabei aber immer nur auf jeweils einem Kanal.

Ein *Local Bus Guardian* muSS die gleichen Funktionen erfüllen können wie ein *Communication Controller*. Statt aber Daten zu senden, überwacht er nur den eigentlichen *Communication Controller* bei seiner Arbeit. Er führt dazu eine eigene Zeitsynchronisation durch. Bei Unstimmigkeiten zwischen dem *Bus Guardian* und dem *Communication Controller* wird dem *Bus Driver* die Sendeerlaubnis entzogen und der Fehler dem *Host* und dem *Communication Controller* gemeldet. Um die Funktion des *Communication Controller* korrekt nachvollziehen zu können, muSS der *Bus Guardian* vom *Host* alle eigentlich an den *Communication Controller* gerichteten Anweisungen ebenfalls, und sogar einige Zeit früher erhalten. Der *Local Bus Guardian* kann auch von einem eigenen (anderen) *Host* Befehle erhalten. Die Hosts müssen dann aber anderweitig untereinander kommunizieren können.

Mit Hilfe eines *Central Bus Guardian* kann eine Busüberwachung auch auSSerhalb der Knoten (also ohne Um-/Aufrüstung) erfolgen. Um auf einzelne Knoten reagieren zu können, muSS ihm ein fester Netzplan zur Verfügung gestellt werden. Es empfiehlt sich die Anordnung des *Central Bus Guardian* am Kreuzungspunkt eines Sterns, da die Überwachung (und etwaige Sperrung) nur pro Inputleitung, nicht pro Knoten festgesetzt werden kann. Mehrere Sicherungsfunktionen stehen zur Verfügung. Generell werden alle Nachrichten dekodiert, anlaysiert und wieder encodiert. Durch das erneute Encodieren werden die Signalpegel regeneriert, was zum besseren Empfang über gröSSere Entfernungen beiträgt. Durch die Analyse können syntaktisch falsche Nachrichten erkannt und als fehlerhaft markiert werden, die weitere Verarbeitung wird aber den Zielknoten überlassen. Mit Hilfe einer eigenen Zeitsynchronisation und dem Netzplan kann auSSerdem das Scheduling[4] der Nachrichten überwacht werden. Dazu arbeitet der *Central Bus Guardian* als selektiver Forward/Filtering Punkt im Netz, der nur die Inputleitung des gerade erlaubten Knotens abhört und weiterleitet, alle anderen hingegen ignoriert.

Der Zentrale *Bus Guardian* ist auch der einzige Schutz gegen einen Netzwerkknoten, der ständig fehlerhafte Daten auf den Bus sendet (im Flexray Jargon: "Babbling Idiot"). Schlägt die Sendekontrolle eines lokalen Buswächters nämlich fehl, werden alle Daten durch falsch synchronisierte Übertragungen überlagert und können so auch die Synchronisation der anderen Knoten stören. Ein *Central Bus Guardian* kann diese Störung jedoch erkennen und diesen Netzwerkzweig abschalten oder ignorieren.

[4]Siehe dazu "Netzwerkszugriffsteuerung", Kapitel 4.2, Seite 16

4 Funktionen

4.1 Kodierung / Dekodierung

Der *Communication Controller* ist für die Kodierung der Nachrichten zuständig. Die vom *Host* übermittelten Nutzdaten werde in Rahmen (Frames) verpackt und mit zusätzlichen Sicherungs- und Steuerungsinformationen versehen. In einem zweiten Schritt werden diese Frames unter Hinzufügung weiterer Steuerdaten in einen Bitstream codiert, der dem *Bus Driver* übergeben wird. Im Gegenzug liefert dieser einen Bitstream zurück, der dann decodiert wird um Frames und Symbole zu erkennen und die extrahierten Nutzdaten zurück zum *Host* zu schicken.

4.1.1 Frames

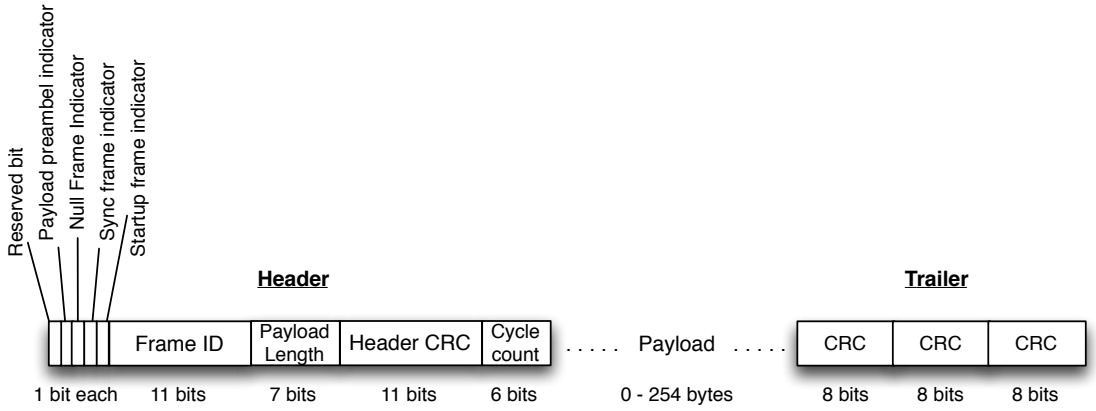

Abbildung 11: Aufbau eines Frames

Reserved bit für spätere Nutzung reserviert, wird bis dahin stets 0 gesetzt

Payload preambel indicator weiSSt auf speziellen Inhalt am Anfang der Payload hin: Message ID[5] (2 Byte bei dynamisch Übertragung) oder Network Management Vector[6] (1-12 Byte bei statische Übertragung)

Null frame indicator zeigt ein Null Frame an, also ein Frame ohne nutzbare Daten (Achtung! Belegung 0 → Null Frame, 1 → normales Frame

Sync frame indicator kündigt ein Sync Frame zur systemweiten Kommunikationssynchronisation an

Startup frame indicator kennzeichnet den Rahmen als Startup Frame, darf nur von *Coldstart* Knoten gesendet werden, immer in Verbindung mit Sync frame indicator

[5] Nachrichtenkennung auf Applikationsebene, für Controller unrelevant
[6] Informationen über den Netzwerkaufbau zur Schedule-Planung durch die Applikationsebene

Frame ID eindeutige Kennzeichnung eines Frames innerhalb eines Kommunikationszyklus; Wertebereich 1-2047

Payload length Payload Länge in Byte geteilt durch 2, konstant bei statischer Übertragung

Header CRC Checksumme über Sync und Startup indicator, Frame ID und Payload length; wird nicht vom Controller berechnet, sondern als Parameter übergeben; konstant bei statischer Übertragung

Cycle count aktueller Wert des Zykluszählers beim Versenden der Nachricht; Wertebereich 0 - 63

Trailer CRC 24 bit CRC über das gesamte Frame

4.1.2 Segmentierung des Bitstreams

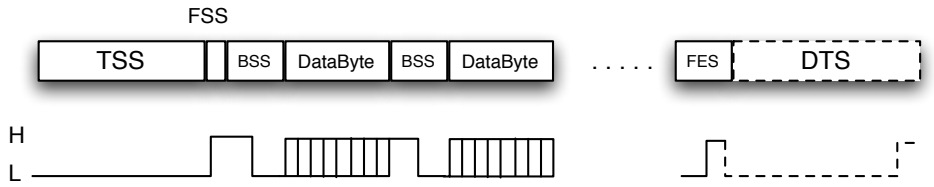

Abbildung 12: Aufbau des Bitstreams zur Frame-Übermittlung

TSS Transmission Start Sequence: festgelegte Zeitdauer LOW-Pegel, um den Übertragungsstart zu kennzeichnen. (Erkannt und dabei z.T. abgeschnitten durch Sternknoten o.ä.)

FSS Frame Start Sequence: 1 Bit (HIGH), zur Verhinderung von Quantisierungsfehlern

BSS Byte Start Sequence: 2 Bit (HIGH LOW), Kennzeichnet Beginn eines Datenbytes. Wird vor jedem weiteren Byte wiederholt. BSS und Daten Byte werden zusammen als Extended Byte Sequence bezeichnet.

DataByte Nutzdaten: byteweise segmentierter Nutzinhalt, also ein kompletter Frame (siehe 4.1.1)

FES Frame End Sequence: 2 Bit (LOW HIGH), markiert Ende eines Frames. Bei statischer Übertragung wird mit dem 2ten Bit die Übertragung beendet.

DTS Dynamic Trailing Sequence: nur bei dynamischer Übertragung. Füllt Zeitslot mit LOW Pegel auf, endet mit einem Bit High.

4.1.3 Majority Voting

Beim Empfangen des Bitstreams wird zur Fehlerreduktion, besonders zur Unterdrückung von Glitches[7], ein Majority Voting durchgeführt. Der Bitstream wird daher nicht direkt eingelesen, sondern Samples entnommen und in einem Sliding Window[8] bewertet. Ist die Mehrheit der Samples HIGH, ist das Outputsignal HIGH. Kurzzeitige Störungen werden so ausgefiltert, allerdings verschiebt sich die Erkennung gewollter Pegelwechsel um einen festen Wert (Delay) entsprechend der Breite des Sliding Window.

4.1.4 Bitsynchronisation/-strobing

Die Byte Start Sequence dient zur Synchronisation der Bitabgrenzungen. Bits werden durch mehrfache Abtastung (Strobing) erkannt. Die Anzahl der Samples pro Bit sollte konstant sein, kann aber durch Übertragungsfehler abweichen. Die fallende Flanke zwischen dem HIGH und LOW Bit der BSS wird genutzt, um den Sample Counter zurückzusetzen. Etwaige Verschiebungen werden so wieder ausgeglichen.

4.1.5 IDLE - Erkennung

Der Buszustand IDLE kann vom Controller nicht direkt vom Bus erkannt werden. Vom *Bus Driver* wird IDLE als HIGH in den Bitstream gegeben, ein Prozess im Controller detektiert dann konstante HIGH - Folgen und wertet diese als IDLE aus. Wird gerade selbst gesendet oder ein eingehendes Frame erkannt, setzt die Detektierung des IDLE Zustands aus.

[7] kurzzeitige, ungewollte Pegelwechsel
[8] Rahmen fester Breite, der sich zeitlich mit dem Signal verschiebt

4.2 Netzwerkzugriffssteuerung (Media Access Control)

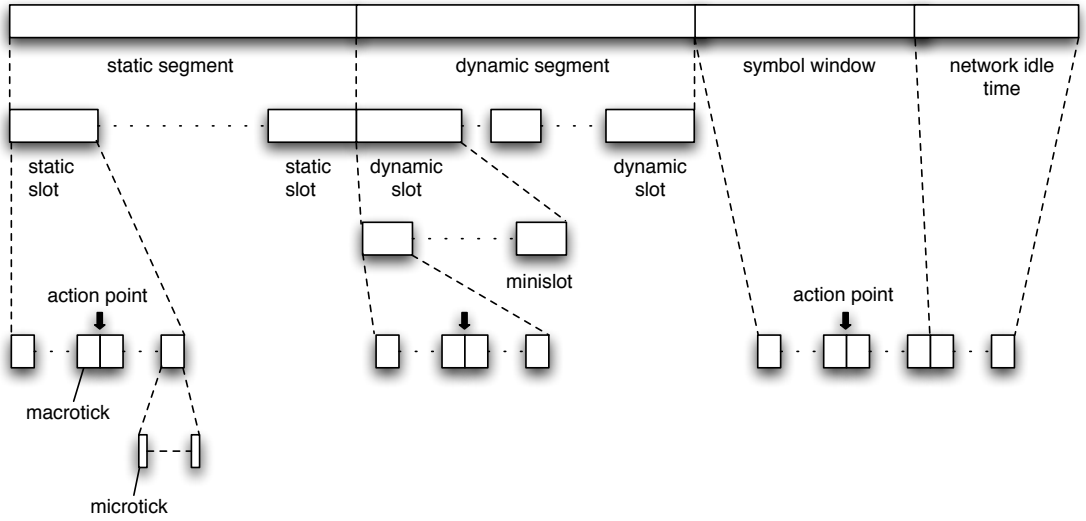

Abbildung 13: Zeitliche Hirarchie innerhalb eines Kommunikationszyklus

Ein Kommunikationszyklus besteht aus 4 aufeinanderfolgenden Abschnitten. Diese reservieren Zeitbereiche für den statischen Übertragungsmodus, den dynamischen Übertragungsmodus, der Übertragung von Symbolen und einer abschlieSSenden Idle Time ohne Netzwerkverkehr. Der statische Bereich setzt sich aus gleichförmigen static slots zusammen, der dynamische Bereich hingegen aus dynamic slots variabler Länge, die aber wiederum aus einer sich unterscheidenden Anzahl an gleichlangen minislots zusammengesetzt sind. Diese, sowie die static slots unterteilen sich weiter in Macroticks und in noch feinere Microticks. Eine genaue Beschreibung der Zeiteinheiten findet sich im Kapitel 4.4 ”Zeitsynchronisation” auf Seite 21.

4.2.1 Statische Nachrichtenübertragung

Die statische Übertragung im static segment des Kommunikationszyklus erfolgt nach einem Time Devision Multiple Access (TDMA) Verfahren. Das Segment besteht aus einer konstanten Anzahl von static slots fester Länge. Sie sind über die auf dem Kanal eindeutige Frame ID fest einzelnen Sendeknoten zugeordnet. Dabei können aber durchaus einem Knoten auch mehrere Frame ID’s zugehören.

Die Übertragung des Frames beginnt erst eine feste Zeit (action point offset) nach Begin des Slots. Endet der Frame, wird noch ein Idle delimiter gesendet und danach der Kanal auf Idle geschaltet, bis die nächste Übertragung startet. Ein Zähler (slot counter), der in allen Knoten synchron arbeitet, bestimmt den gerade aktiven Slot und somit die gerade zum Senden zugelassene Frame ID.

Die Zuordnung der Knoten zu den Frame IDs erfolgt über die Applikationsschicht. Im statischen Segment ist jeder Slot genau einem Knoten zugeordnet, wobei aber ein

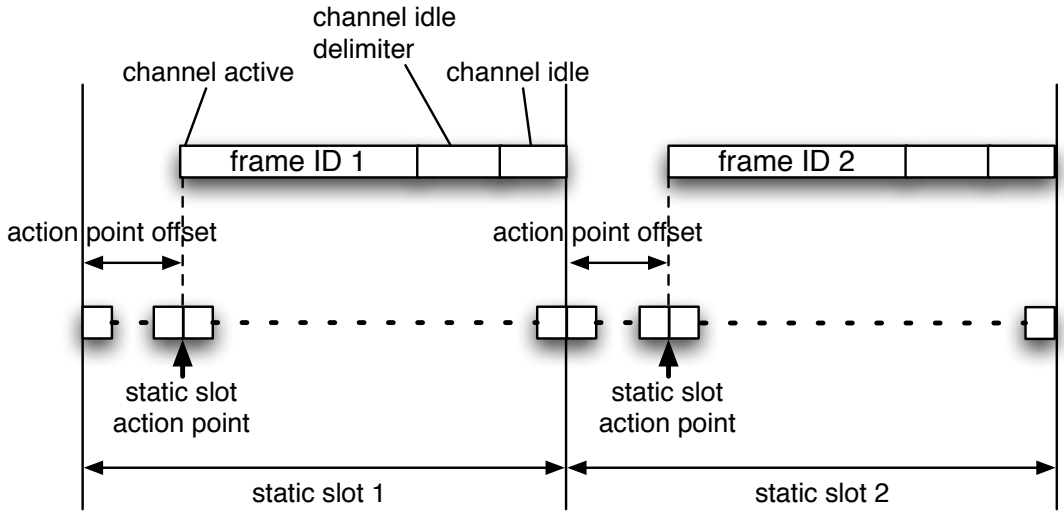

Abbildung 14: Ablauf bei statische Übertragung

Knoten auch mehrere Slots belegen kann. Hat der Knoten keine Daten zu übertragen, sendet er ein null frame.

4.2.2 Dynamische Nachrichtenübertragung

Im Segment zur dynamischen Übertragung können Daten variabler Länge gesendet werden. Damit kann eine gröSSere Datenmenge übertragen werden, wenn das gesamte Segment zum Beispiel nur von einem Knoten zum Senden verwendet wird. Dafür kann keine Garantie zur Bereitstellung eines Slots gegeben werden. Der slot counter[9] bestimmt die gerade zum Senden zugelassene Frame ID, der Knoten mit Frame ID = 1 erhält also als erstes die Gelegenheit zum Senden. Somit wird indirekt eine prioritätsgesteuerte Vergabe von Senderechten realisiert.

Auch im dynamischen Segment beginnt die Übertragung erst nach einem Offset. Zur besseren Abgrenzung vom statischen Segment wird beim ersten slot hier immer der gröSSere der beiden Offsets (statisch oder dynamisch) verwendet. Der dynamic slot passt sich der Länge der übertragenen Daten an. Werden keine Daten gesendet, ist er mindestens einen minislot lang. Nach erfolgreicher Übertragung eines Frames wird mittels Idle delimiter und dem Idle Zustand bis zum Ende des nächsten minislots aufgefüllt.

Auch hier erfolgt die Zuordnung der Frame IDs zu den Knoten über die Applikationsschicht. Stehen keine Daten zum Senden bereit, wird kein Frame gesendet. Anders als im statischen Segment ist hier ein Slot-Multiplexing erlaubt, es können sich also mehrere Knoten einen Slot teilen (zB gerader/ungerader Zykluszähler). Die Organisation dieses Multiplexings obliegt wiederum der Applikationsschicht.

[9] Der slot counter zählt hier die dynamischen slots, nicht die konstanten minislots

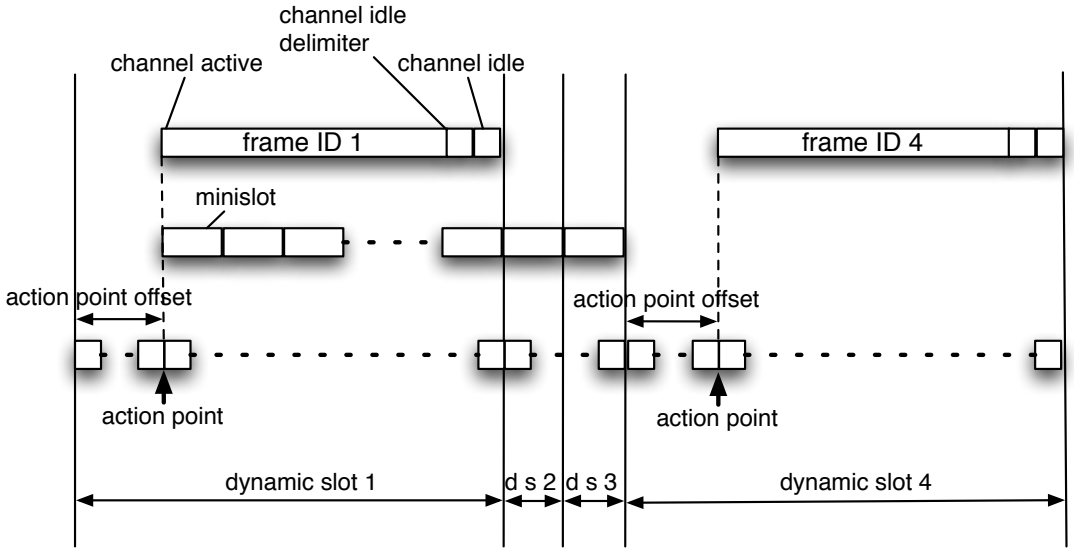

Abbildung 15: Ablauf bei dynamischer Übertragung

4.2.3 Fenster zur Symbolübertragung

Im symbol window werden spezielle Symbole, also einzelne Steuersignale, übertragen. Hierzu zählen zum Beispiel WakeUp Signale. Die Übertragung erfolgt in gleicher Weise wie ein Slot im statischen Segment. Das Symbolfenster wird auch bei sonst abgeschalteter Kommunikation überwacht und direkt durch den *Bus Driver* ausgewertet. Im aktiven Zustand werden Symbole zusätzlich zum *Communication Controller* weitergeleitet.

4.3 Initialisierungsphase

Die normale Übertragung im *Flexray*netzwerk benötigt einige Parameter, die in allen Knoten synchron gehalten werden müssen beziehungsweise zumindest einer vorherigen Absprache bedürfen. Dazu dient zum einen der Zeitsynchronisationsprozess (Kapitel 4.4, Seite 21), aber insbesondere auch eine komplexe Startphase des Netzes.

4.3.1 WakeUp

Die Knoten eines Netzes können sich in Sleep beziehungsweise StandBy Zuständen befinden, müssen also gegebenenfalls erst aufgeweckt werden. Dies geschieht durch ein WakeUp Symbol (siehe auch Kapitel 3.3, Seite 9), das von einem als WakeUp-Knoten gekennzeichneten Knoten gesendet wird. Ein Knoten weckt hierbei immer nur einen Kanal. Erst der als nächste aufwachende WakeUp-Knoten weckt daraufhin den zweiten Kanal. Vor dem Senden wird stets abgewartet ob wirklich keine Kommunikation auf dem Kanal stattfindet, also wirklich alle Knoten sich im Sleep/StandBy Modus befinden. Sollten zwei Knoten gleichzeitig versuchen den Bus aufzuwecken, tritt der zurück, der die Kollision zuerst bemerkt und wartet anschlieSSend ob der andere Weckaufruf erfolgreich ist. Die Signale selbst sind kollisionsresistent, ein überlagertes, zweites Wecksignal schränkt also die Funktion des ersten nicht ein.

4.3.2 StartUp

Das Starten des Netzwerkverkehrs nach dem Aufwecken aller Knoten kann nur von als Coldstart-Knoten bezeichneten Knoten vorgenommen werden (üblicherweise mind. 3 pro Bus). Dies müssen nicht zwingend dieselben sein, die auch als WakeUp-Knoten genutzt werden.

Der initiierende Knoten (leading coldstart node) sendet ein Kollisions-Vermeidungs-Signal CAS an das Netz, welches ihm 4 Zyklen alleiniges Senderecht garantiert. Danach folgen die anderen Coldstart-Knoten (following coldstart nodes), daraufhin alle anderen Knoten, die mindestens von 2 Coldstart-Knoten ein StartUp Frame erhalten müssen.

Knoten 1 (leading coldstart node): Nach dem Senden des CAS Signals beginnt Knoten 1 mit dem Übermitteln von StartUp Frames, gleichzeitig beginnt der erste Zyklus. Die ersten 4 Zyklen werden zum detektieren von Kollisionen benutzt. Werden andere CAS Signale oder StartUp Frames empfangen, wird der Versuch abgebrochen. Sobald zwei gültige und zeitsynchrone StartUp Frames eines anderen Knotens empfangen werden, geht Knoten 1 in den normalen Zustand über.

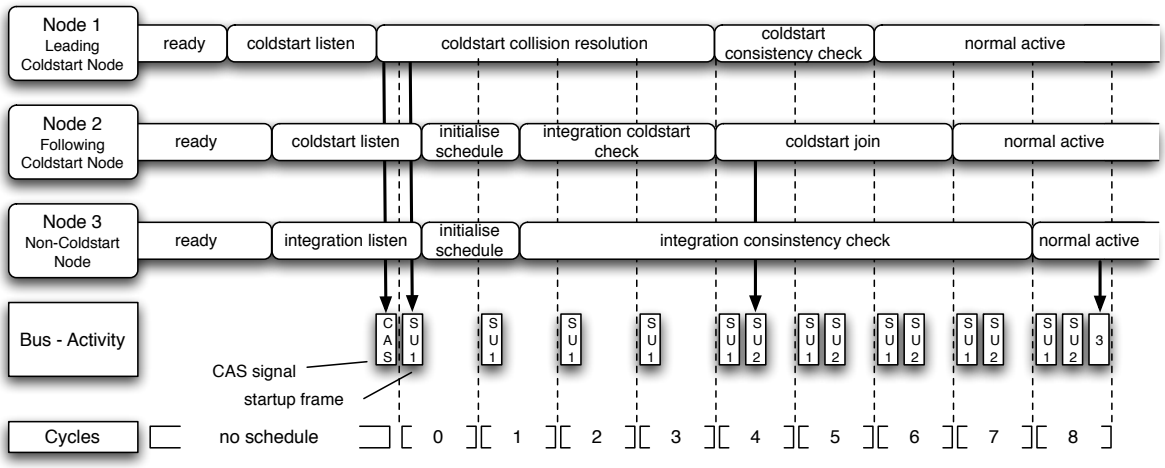

Abbildung 16: Initialisierung der Kommunikation (Coldstart)

Knoten 2 (following coldstart node): Empfängt Knoten 2 mind. zwei StartUp Frames, fängt er an sich auf diese zu synchronisieren. An den nächsten 2 Frames wird diese Synchronisation verifiziert und am Ende der 4 Zyklen Wartezeit nach dem CAS Symbol beginnt Knoten 2 mit dem Senden von eigenen StartUp Frames. Um Knoten 1 genügend Frames zur Verfügung zu stellen und um nicht vor Knoten 1 aktiv zu werden, wartet Knoten 2 noch drei weitere Zyklen ab. Ist die Synchronität noch immer gewährleistet und werden immernoch Frames von Knoten 1 empfangen, wechselt Knoten 2 in den normalen, aktiven Zustand.

Knoten 3 (non-coldstart node): Nach dem Empfang von 2 StartUp Frames beginnt auch Knoten 3 mit der Synchronisation auf diese. Sobald danach für 4 Zyklen korrekte und synchrone StartUp Frames von mind. 2 unterschiedlichen Coldstart Knoten empfangen werden, wechselt auch Knoten 3 in den aktiven Zustand und beginnt normale Frames zu versenden.

Empfängt ein Knoten wider Erwarten in entsprechender Zeit kein Frame, wird der Startvorgang abgebrochen. Coldstart Knoten versuchen dann einen neuen Start nach einer festgelegten Pause, im Falle dass noch nicht alle Knoten bereit waren.
Anhand der eingehenden Frames wird bei der Synchronisation insbesondere der Zykluszähler für die Zeitplanung der folgenden Übertragungen entsprechend initialisiert.

4.4 Zeitsynchronisation

Jeder Knoten hat einen eigenen Oszillator zur Zeitgenerierung. Trotzdem weichen schon nach kurzer Zeit die Taktsignale voneinander ab, vor allem auf Grund von Fertigungsunterschieden, elektromagnetischen Störungen und Wärmeunterschieden. Daher übernimmt der *Communication Controller* die Synchronisation der Zeitbasis anhand empfangener Nachrichten auf dem Bus.

4.4.1 Hierarchie der Zeiteinheiten

Das Synchronisationsprotokoll verwendet drei hierarchische Zeiteinheiten:

Microtick: kleinste Zeiteinheit, basiert auf lokalen Oszillator, kann sich pro Knoten unterscheiden

Macrotick: besteht aus ganzzahliger Anzahl Microticks, in allen Knoten gleich lang, erste synchronisierte Zeiteinheit

Cycle: besteht aus ganzzahliger Anzahl Macroticks, in allen Knoten gleich lang, alle Knoten befinden sich im gleichen (nummerierten) Cycle

Die Anzahl der Microticks in einem Macrotick kann von Knoten zu Knoten, aber auch von einem Macrotick zum nächsten variieren. Zwischen Macrotick und Cycle werden je nach Übertragungsmodus noch weitere logische Unterteilungen vorgenommen.

4.4.2 Korrekturen

Zwei Arten der Korrektur werden angewendet, um alle Knoten synchron zu halten, **offset correction** und **rate correction**. Die Korrektur der Frequenz (rate correction) erfolgt durch Einfügen oder Entfernen von Microticks pro Macrotick. Diese Anpassung erfolgt über den gesamten Zyklus hinweg. Die Korrektur der Verschiebung (offset correction) erfolgt hingegen durch Einfügen von zusätzlichen Microticks am Ende des Zyklus in der network idle time.
Die zu diesen Korrekturen notwendigen Messungen werden an im statischen Segment übertragenen sync frames vorgenommen, die von speziell als Sync-Knoten gekennzeichneten Knoten ausgesendet werden. Die Messungen basieren jeweils auf dem Unterschied zwischen der mit den bisherigen Werten berechneten Eintreffzeit eines sync frames und der gemessenen.

Der Offset wird zwar in jedem Zyklus berechnet, aber nur in jedem zweiten (ungerader Zykluszähler) angewendet. Die rate correction hingegen wird über alle Zyklen hinweg angewendet, aber nur alle zwei Zyklen neu berechnet. Die Berechnung erfolgt anhand mehrerer Messwerte über einen Midpoint Algorithmus. Extremwerte am oberen und unteren Rand werden verworfen, als Korrektur dann der Mittelwert der verbliebenen Werte benutzt.

Befinden sich die Korrekturwerte auSSerhalb gewisser akzeptabler Grenzen, wird nichtmehr korrigiert sondern ein Fehler zum *Host* gemeldet und die Übertragung abgebrochen.

5 Fazit

Mit *Flexray* wurde ein neues Kommunikationsprotokoll entwickelt, das sich sehr schwierigen Anforderungen stellen muSS. Trotz neuem Ansatz in vielen Gebieten, muSS sich wohl erst in einer ausführlichen Testphase zeigen ob diese auch wirklich alle erfüllt werden können.

Der neue Ansatz von *Flexray* besteht hauptsächlich darin, bisherige Vorteile verschiedener Bussysteme zu verbinden. Durch TDMA - Verfahren wird eine determinierte Übertragung realisiert, die zu garantierten Latenzzeiten und gut berechenbaren Übertragungseigenschaften führt. Trotzdem werden durch das dynamische Segment auch grösSere Datenmengen übertragen und aperiodische Nachrichten müssen sich nicht den Übertragungsraum mit dem dauerhaften Datenstrom teilen. Zusätzlich wird die Möglichkeit zum Powermanagement direkt auf Busebene gegeben.

Leider verlagert *Flexray* aber viele Probleme nur auf andere Ebene des Gesamtsystems. Eine Aussage, ob besagte Probleme damit durch *Flexray* gelöst werden ist also schwierig.

So findet zum Beispiel eine Standardisierung der rein physikalisch / mechanischen Bauteile (Kabel, Steckerverbindungen etc.) nicht statt, obwohl für ein "Open System" aus Anwendersicht sehr wünschenswert.
Am anderen Ende der Funktionskette werden viele Funktionen auf die Applikationsebene ausgelagert, die eigentlich zum Übertragungssystem gehören. So zum Beispiel das gesamte Netzwerkmanagment und die Adressierung.

Da *Flexray* also nur einen sehr schmalen Bereich des gesamten Übertragungsweges einnimmt, fällt eine Einordnung in das ISO/OSI Schichtenmodell recht schwer. Funktionen der Schicht 3 sind nicht zu erkennen, also keine Adressierung, keine Wegewahl, keine Punkt-zu-Punkt Verbindung. Die gesamte Funktionalität findet sich also zwischen Schicht 1 und 2. Die zweifache Kodierung in Frames und in einen Bitstream legt auSSerdem eine weitere Unterteilung ähnlich dem IEEE Zusatz in Logical Link Control und Media Access Control nahe. In Schicht 1 findet die Leitungskodierung statt, die sich in der Spezifikation allerdings nur als Vorschlag findet. Ähnlich wie die auch in Schicht 1 definierten physikalischen Übertragungsmedien und Steckverbindungen.

Die Anforderung an Fehlerfreiheit (fault-free) und Fehlertoleranz (fault-tolerance) werden nur innerhalb des *Flexray* -Systems erfüllt. Toleranz heiSSt also, das das System weiterläuft, obwohl Busfehler vorliegen. Das Problem fehlender Daten hat die Applikationsschicht zu lösen. Fehlerfreiheit wird nur bei korrekter Funktion aller Knoten garantiert, eine Fehlfunktion innerhalb eines Flexrayknotens kann nur ein zentrale Buswächter lösen, in dem er den kompletten Buszweig abschaltet. Eine Kollisiondetekierung oder -verhinderung vor jedem Senden findet nicht statt.

Der *Bus Guardian* wurde erst im Nachtrag Ende 2005 spezifiziert. Scheinbar konnten die Anforderungen nicht so einfach gelöst werden wie anfangs geplant, ist er doch deutlich komplexer als bisher vorgesehen. Als lokaler Buswächter muSS er dieselben Funktionen wie der *Communikation Controller* parallel ausführen, um diesen zu kontrol-

lieren. Das bedeutet redundante Hardware und einen Schutz nur auf Sendebasis. Der zentrale Buswächter ist wesentlich mächtiger und effektiver, widerspricht damit aber eigentlich dem dezentralen Grundansatz.

Desweiteren bleiben für mich persönlich zwei offene Fragen:

- Trotz ausführlicher Beschreibung des StartUp Prozesses bleibt unklar, wie und wann die initiale Zuteilung der Frame IDs erfolgt. Zwar soll die Zuordnung durch die Applikationsschicht erfolgen, doch mindestens beim ersten Startup kann dies noch nicht geschehen. → Vorkonfiguration?
- Die Integration von Non-Coldstart Knoten benötigt StartUp-Frames zur Synchronisierung. Die geforderte Integrierbarkeit von Knoten in ein laufendes Netz fordert also die ständige Versendung von StartUp Frames und somit Belegung von Übertragungsbandbreite.

Diese Fragen und alle anderen im Fazit erwähnten Probleme spiegeln allerdings nur meinen persönlichen Eindruck nach theoretischem Studium der Spezifikationen wider und stellen natürlich keine objektive Bewertung dar. Dazu wäre auf jeden Fall die Testphase des Systems noch abzuwarten.

6

Abbildungsverzeichnis

7 Quellenverzeichnis

Die in diesem Dokument enthalten Informationen stammen aus den hier aufgelisteten Quellen. Auf Verweise im Text wurde verzichtet, da nirgends wörtlich zitiert wird. Alle Grafiken sind vom Autor erstellt, benötigen also keine Quellenangabe.

1. FlexRay Consortium, *FlexRay Communications System Protocol Specification* Version(2.1 Rev. A), (www.flexray.com, 15. Dez. 2005)
2. FlexRay Consortium, *FlexRay Requirements Specification* Version(2.1), (www.flexray.com, 19. Dez. 2005)
3. FlexRay Consortium, *FlexRay Communications System Electrical Physical Layer Specification* Version(2.1 Rev. A), (www.flexray.com, Dez. 2005)
4. FlexRay Consortium, *Preliminary Node-Local Bus Guardian Specification* Version(2.0.9), (www.flexray.com, 15. Dez. 2005)
5. FlexRay Consortium, *Preliminary Central Bus Guardian Specification* Version(2.0.9), (www.flexray.com, Mai 2005)
6. Claas Bracklo, *FlexRay International Seminar - Consortium Information Update* Presentation Slides, (www.flexray.com, 4. Juni 2003)
7. Bernd Elend, *FlexRay International Workshop - Flexray The electrical physical layer* Presentation Slides, (www.flexray.com, 4. März 2003)