Deutsches Zentrum
für Luft- und Raumfahrt
Institut für Verkehrsführung
und Fahrzeugsteuerung

Fachhochschule
Braunschweig / Wolfenbüttel
Fachbereich Elektrotechnik

Diplomarbeit

Thema:

Entwicklung von Hardware- und Softwarekomponenten für ein DSP – System zur Anbindung eines Zugfahrzeugrechners an ein Bahnsimulationslabor über Eurobalisen

Bearbeiter:

Name	Martin Willms
Matrikel Nr.:	20064453

Prüfer:

Erstprüfer:	Prof. Dr. Dipl. Ing. K. H. Kraft
Zweitprüfer:	Dipl. Ing. V. Knollmann

Datum:

13.01.2006

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich diese Arbeit ohne Hilfe dritter und nur unter Verwendung der im Quellenverzeichnis angegebenen Hilfsmittel erstellt habe.

Datum

Unterschrift

1 Inhalt:

Abbildungen:

Tabellen:

Formeln:

2 Einführung

2.1 Zweck dieses Dokuments

Dieses Dokument ist die schriftliche Ausarbeitung der Diplomarbeit. Die Arbeit wurde während eines Praktikums im deutschen Zentrum für Luft- und Raumfahrt (DLR) am Standort Braunschweig im Institut für Verkehrsführung und Fahrzeugsteuerung erstellt.

Es beschreibt eine mögliche Realisierung einer Anbindung von realen Zugfahrzeugrechnern an das RailSiTe (Rail Simulation and Testing) Labor des DLR mittels eines DSP (digitaler Signalprozessor) und Peripheriekomponenten.

Die Schnittstelle bildet dabei eine Funkverbindung über eine Eurobalise, die im neuen europäischen Zugleit- und Sicherungssystem ETCS (European Train Control System) zur punktuellen Zugbeeinflussung (PZB) genutzt wird.

Ziel der Arbeit ist die Möglichkeit, das Fahrzeuggerät auf einer virtuellen Strecke, die durch das RailSiTe Labor simuliert wird, fahren zu lassen. Auf diese Weise können die Fahrzeugrechner unter Laborbedingungen auf Interoperabilität getestet werden. Durch den festen Aufbau innerhalb eines Labors besteht dabei die Möglichkeit, zielsicher nach Fehlerquellen suchen zu können, ohne dafür große Teststrecken und echte Fahrzeuge ausrüsten zu müssen.

Hauptaugenmerk liegt auf der Implementation der Algorhythmen zur Kodierung der bei der Übertragung genutzten Telegramme. Dazu wird der verwendetet Kodierungsalgorhythmus im Kapitel 4 zuerst kurz beschrieben, um anschließend ein Konzept zur Implementation aufzuzeigen.

Im Kapitel 5 wird eine Recherche über mögliche Übertragungsfehlermodelle durchgeführt. Dabei wird besonderer Wert auf die für die in ETCS verwendete Kanalkodierung wichtigen Fehlermodelle gelegt. Anschließend ist auch hier ein Konzept zur Implementation der Fehlermodelle gegeben.

Das Kapitel 6 beschreibt die Anforderungen und Umsetzung eines Diagnosekanals, über den die DSP - Software im laufenden Betrieb überwacht und direkt auf Funktionen des DSP zugegriffen werden kann.

Im Kapitel 7 wird schließlich der Aufbau eines FSK Modulators (engl. Frequency Shift Keying = Frequenzumtastung) beschrieben, der für die Übertragung der Datentelegramme über die Funkschnittstelle der Balise notwendig ist.

2.2 Der Betrieb

Das deutsche Zentrum für Luft- und Raumfahrt (DLR) ist das nationale deutsche Zentrum für die Forschung in den Themengebieten Luftfahrt, Raumfahrt und Verkehr. Neben seinem Stammsitz in Köln betreibt das DLR sieben weitere Standorte in Deutschland.

Dabei wird neben der Luft- und Raumfahrt auch in anderen Themengebieten geforscht. Hier sei insbesondere der Teilbereich Verkehr herausgehoben, der die Erkenntnisse des DLR aus der Luft- und Raumfahrtforschung auf den erdgebundenen Verkehr übertragen soll.

Innerhalb des Teilbereiches Verkehr des DLR befindet sich das Institut für Verkehrsführung und Fahrzeugtechnik am Standort Braunschweig. Dort werden Assistenzsysteme und Simulationswerkzeuge sowohl für den straßen-, als auch für den schienengebundenen Verkehr erforscht.

Diese Arbeit ist im Projekt RailSiTe (Rail Simulation and Testing) entstanden. Dabei handelt es sich um ein Testlabor, in dem Komponenten der Zuleit- und Sicherungstechnik auf funktionelle Kompatibilität getestet werden können. Eine genauere Beschreibung des RailSiTe Labors ist in Kapitel 3.2 gegeben.

3 Rahmenbedingungen

3.1 Einführung in ETCS (European Train Control System)

Da diese Arbeit die Anbindung eines ETCS Zugfahrzeugrechners an das Simulationslabor RailSiTe des DLR beschreibt, soll in diesem Abschnitt eine kurze Einführung in dieses System gegeben werden.

Die ETCS Spezifikation beschreibt ein einheitliches Zugleit- und Sicherungssystem für den gesamteuropäischen Bahnverkehr. Die Motivation hinter der Entwicklung dieses Systems ist die unglaubliche Vielfalt an Zugleit- und Sicherungssystemen innerhalb Europas, die größtenteils zueinander inkompatibel sind. Aus diesem Grund ist inneuropäischer Zugverkehr teuer und schwer zu realisieren, da die grenzüberfahrenden Fahrzeuge mit allen Zugleit- und Sicherungssystemen ausgestattet sein müssen, die in den bereisten Ländern notwendig sind. Notfalls müssen an den Landesgrenzen Fahrzeuge ausgewechselt werden, was zu Verspätungen führt und dadurch ebenfalls Kosten verursacht. Abbildung 1 zeigt nur eine kleine Übersicht über die Vielzahl der derzeit eingesetzten Systeme.

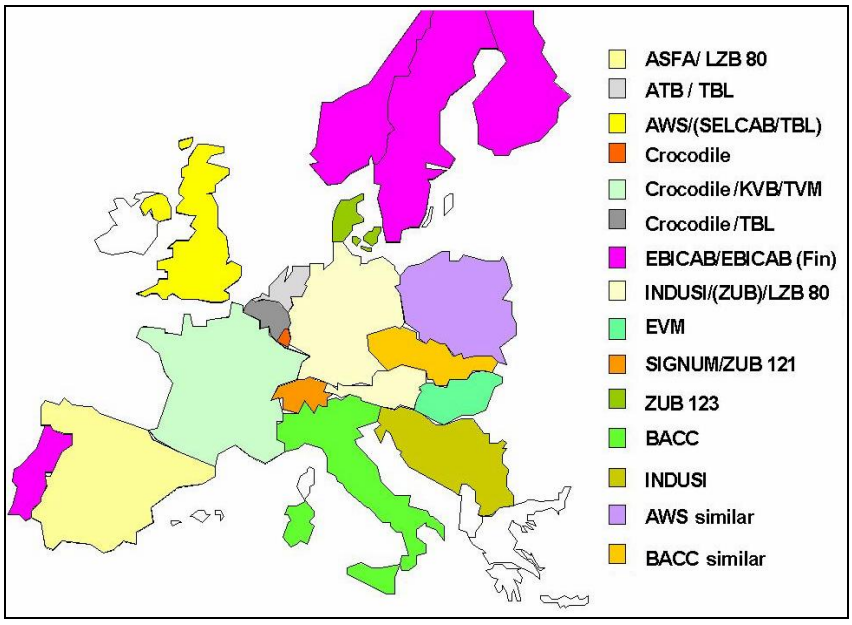

Abbildung 1: derzeit eingesetzte Systeme der Zugleit- und Sicherungstechnik (Quelle: [9])

Um diese Systeme in einem einheitlichen europäischen System zusammenzufassen und so europäischen Zugverkehr zu vereinfachen, haben sich die namhaftesten europäischen Firmen der Zugleit- und Sicherungstechnik zum EUROSIG Konsortium zusammengeschlossen, das dieses neue System spezifizieren und umsetzen soll.

Grundsätzlich besteht das ETCS System aus standardisierten Komponenten, die zwar von verschiedenen Herstellern gefertigt, aber vor Inbetriebnahme zertifiziert und freigegeben werden müssen, um eine Kompatibilität der Komponenten der verschiedenen Hersteller zu errei-

chen.. Die für diese Arbeit wichtigsten Komponenten werden in der nachfolgenden Tabelle näher erläutert.

Tabelle 1: Komponenten des ETC Systems

Name	Beschreibung
Eurobalise	Eine Balise ist ein Streckenseitiger Transponder, der Telegramme mit Streckeninformationen über eine Funkstrecke an ein Fahrzeug übermittelt. Die Kommunikation zwischen Balise und Fahrzeug wird dabei automatisch bei überfahren der Balise durch das Fahrzeug aktiviert. Dazu wird vom Fahrzeug ein Telepowering Signal übertragen, das die Balise mit Energie versorgt, wenn sich das Fahrzeug über einer Balise befindet. Es handelt sich hierbei also um ein Mittel zur punktförmigen Zugbeeinflussung. Die Balisen können sowohl als Speicher mit festem Dateninhalt, als auch als Transparentbalisen ausgelegt werden, bei denen der Dateninhalt während des Betriebs frei geändert werden kann. Abbildung 2 zeigt eine solche Eurobalise der Firma Siemens.  **Abbildung 2: Eurobalise**
Euroloop	Der Euroloop ist als Erweiterung der Eurobalise zur linienförmigen Zugbeeinflussung gedacht. Er kann, anders als die Eurobalise, beliebige Daten über größere Strecken direkt an ein Fahrzeug senden. Dabei bleibt der Datenstrom im gesamten Bereich des Euroloop bestehen. Der Euroloop kann z.B. überall dort sinnvoll eingesetzt werden, wo linienförmige Zugbeeinflussung notwendig, aber über das in ETCS verwendete GSM-R (engl. Global System for Mobile Communication – Railway) System nicht möglich ist.

Name	Beschreibung
European Vital Computer (EVC)	Bei dem EVC handelt es sich um den bei ETCS verwendeten Fahrzeugrechner. Er wird einheitlich in allen ETCS konformen Fahrzeugen verwendet.  **Abbildung 3: European Vital Computer (EVC)**
Radio Block Center (RBC)	Das Radio Block Center bietet die streckenseitige Zentrale, in der alle Zug- und Streckendaten für einen definierten Streckenabschnitt zusammenlaufen und ausgewertet werden. Das RBC vergibt für alle ihm unterstellten Züge die nötigen Movement Authorities (MA, eng. Fahrerlaubnis) und überwacht die Position aller Züge, um Konflikte auf der Strecke zu vermeiden. In den ETCS Levels 2 und 3 wird eine Kommunikation über GSM-R betrieben, die eine durchgehende Ortung der Züge, und dadurch sehr viel kleinere Blockabschnitte, bis hin zu Moving Blocks erlaubt.
Eurocab	Das Eurocab ist ein standardisiertes Fahrerpult, das in allen ETCS konformen Fahrzeugen das gleiche Aussehen und die gleiche Funktionalität besitzt. Dadurch soll den Lokführern der Umstieg zwischen verschiedenen Fahrzeugtypen stark erleichtert werden. Es verfügt außerdem über standardisierte Anzeigeelemente.  **Abbildung 4: Driver Machine Interface (DMI)**
Lineside Electronic Unit (LEU)	Die LEU ist eine elektronische Einrichtung, die streckenseitig das Ansteuern der Eurobalisen, oder auch Euroloops übernimmt. Dabei richtet es seine Daten nach den aktuellen Stati der angeschlossenen Signale.

Das ETCS System soll eine konsequente Weiterentwicklung bereits bestehender Systeme sein. Vordergründig soll es die möglichen Geschwindigkeiten und Auslastungen auf den Strecken weiter steigern. Trotzdem soll das System auch auf minder stark ausgelasteten Strecken zu erschwinglichen Preisen realisierbar sein.

Aus diesem Grund ist das ETCS System in vier so genannte ETCS Level unterteilt. Diese ETCS Level beschreiben die verschiedenen Ausbau- und Komplexitätsstufen des ETCS Systems. Dabei entspricht der ETCS Level 0 einer einfachen Sichtfahrt mit voller Verantwortlichkeit des Lokführers, wohingegen der ETCS Level 3 bereits so genante „Moving Blocks“ (engl. bewegende Blöcke) als Abschnittsicherung benutzt, bei denen eine belegter Streckenabschnitt immer nur so lang ist, wie der Zug, der auf der Strecke fährt und der Lokführer nur noch überwachende Funktion besitzt.

Auch wenn die in dieser Arbeit beschriebene Anbindung von Balisen hauptsächlich für den ETCS Level 1 sinnvoll ist, sollen nachfolgend die vier ETCS Level kurz erläutert werden. Genauere Beschreibungen des ETCS Systems finden sich in [9].

3.1.1 Level 0

Der ETCS Level 0 stellt praktisch eine Sichtfahrt dar. Dabei besitzt der Lokführer die volle Kontrolle und Verantwortung für sein Fahrzeug. Der Fahrzeugrechner des ETCS Systems übernimmt nur eine Höchstgeschwindigkeitsüberwachung. Alle Meldungen der Strecke erfolgen dabei ausschließlich über optische Signale, oder einzelne Balisen auf der Strecke, die Maximalgeschwindigkeiten vorgeben. Mögliche weitere Quellen für gültige Geschwindigkeitsprofile können feste Geschwindigkeitsgrenzen, z.B. im Rangierbetrieb sein.

Damit eignet sich der ETCS Level 0 ausschließlich für sehr schwach frequentierte Strecken, auf denen die Geschwindigkeiten so gering sind, dass ein Fahren auf Sicht noch sicher möglich ist, oder für den Rangierbetrieb innerhalb betrieblicher Einrichtungen.

Der ETCS Level 0 kann ebenfalls als Rückfallebene benutzt werden, falls Komponenten der höheren ETCS Level versagen.

3.1.2 Level 1

Der Level 1 Betrieb des ETCS gleicht sehr stark einer punktförmigen Zugbeeinflussung (PZB), wie z.B. Indusi. Dabei werden sowohl Streckendaten, als auch Geschwindigkeitsprofile und Positionsdaten punktförmig über Eurobalisen von der Strecke an das Fahrzeug übermittelt.

Der Fahrzeugrechner muss dann selbsttätig aus allen gesammelten Informationen ein für ihn passendes Streckenprofil berechnen. Dabei übernimmt der Fahrzeugrechner zusätzlich zur Überwachung der Höchstgeschwindigkeit die Überwachung der Position.

Er sorgt dafür, dass das Fahrzeug immer am nächsten Hindernis (Signal) zum stehen kommt und berechnet zu diesem Zweck Beschleunigungs- und Geschwindigkeitsprofile. Der Lokführer hat in diesem Fall nicht mehr die volle Kontrolle über sein Fahrzeug. Sollte der Lokführer z.B. ein Geschwindigkeitsprofil nicht einhalten, oder Bremspunkte überfahren, so kann der Fahrzeugrechner aktiv durch Zwangsbremsungen in den Fahrablauf eingreifen.

Die Abschnittssicherung erfolgt weiterhin durch Blockabschnitte, die durch Achszähler Gleisfreimeldungen an die Stellwerke geben. Aus diesem Grund ist die Auslastung auf den Strecken immer noch gering.

Durch die Übertragung von Datentelegrammen mit hohen Datenraten können im Gegensatz zu Indusi in diesem Level viele Zusatzinformationen, wie z.B. Klartextmeldungen von der Strecke an den Lokführer übertragen werden. Das ermöglicht trotz Blockabschnittssicherung viele Zusatzfunktionen gegenüber älteren Systemen.

3.1.3 Level 2

Der ETCS Level 2 gleicht einer linienförmigen Zugbeeinflussung (LZB) mit kontinuierlicher Kommunikation zwischen Fahrzeug (EVC) und Strecke (RBC). Die Daten werden dabei in erster Linie über eine Funkverbindung im GSM-R Netz übertragen. In Bereichen, in denen das GSM - R keine Netzabdeckung bietet (z.B. Tunnels), werden zusätzlich Linienleiter in Form von Euroloops eingesetzt.

Durch den kontinuierlichen Datenaustausch zwischen Strecke und Fahrzeug ist sowohl die Strecke jederzeit über die Fahrzeugposition, als auch das Fahrzeug immer über den Status der vor ihm liegenden Strecke informiert. Dadurch entfallen auf Fahrzeugseite unnötige Brems-

eingriffe und die Strecke kann mit höheren Geschwindigkeiten und Auslastungen betrieben werden.

Trotz der kontinuierlichen Positionsbestimmung wird die Abschnittsfreimeldung immer noch durch Blockabschnitte realisiert. Diese können jetzt aber gegenüber ETCS Level 1 kleiner ausfallen.

3.1.4 Level 3

Der ETCS Level 3 ist eine Erweiterung des ETCS Level 2. Dabei werden allerdings die Blockabschnitte durch so genannte Moving Blocks ersetzt. Diese Moving Blocks sind nur wenig größer als der Zug und bewegen sich auf der Strecke mit dem Fahrzeug.

Ein Zug belegt also immer nur genau den Streckenabschnitt, den er auch physikalisch gerade befährt. Dieses Vorgehen bietet die maximal mögliche Streckenauslastung, da zwei Züge mit einem gewissen Sicherheitsabstand praktisch direkt hintereinander fahren können.

Allerdings muss zusätzlich zu ETCS Level 2 weiterer Aufwand betrieben werden. So muss in jedem Zug eine Integritätsprüfung stattfinden, da die Strecke auf Grund des Fehlens von Achszählern nicht mehr feststellen kann, ob ein Zug einen Abschnitt komplett verlassen hat, oder ob möglicherweise einzelne Waggons des Zuges auf der Strecke zurückgeblieben sind.

Auch die Streckenseite muss komplizierte Berechnungen zur Größe der von einem Zug eingenommenen Streckenabschnitte und der daraus resultierenden Gleisfreimeldungen machen, da die Größe und Position der Blöcke in diesem Fall nicht mehr fest vorgegeben, sondern durch Länge, Geschwindigkeit und Position eines Fahrzeugs auf der Strecke definiert sind.

Ohne eine echtzeitfähige Kommunikation zu jedem Fahrzeug auf einem Streckenteilstück, das von einem RBC überwacht wird, ist dieser Level praktisch nicht realisierbar.

3.2 Systemarchitektur

Das RailSiTe - Labor der DLR ist eine Testumgebung zum Kompatibilitätstest allgemeiner Komponenten der Zugleit- und Sicherungstechnik. Es legt besonderen Wert auf die Möglichkeit, Komponenten des neuen einheitlichen europäischen Zugleit- und Sicherungssystems ETCS zu prüfen.

Dabei ist es dem Labor möglich, sämtliche Komponenten, vom Fahrzeugrechner bis zum Stellwerk zu simulieren. Zu diesem Zweck ist das Labor aus mehreren Rechnern aufgebaut, die jeweils eine Teilkomponente der Zugleit- und Sicherungstechnik darstellen. So gibt es z.B. eigene Rechner zur Simulation der Strecken, der BTMs (engl. Balise Transmission Module), der Züge und der Stellwerke. Alle Rechner kommunizieren untereinander über ein fest definiertes Protokoll.

Durch diesen modularen Aufbau ist es möglich, einzelne Komponenten des Labors durch echte Komponenten der Zugleit- und Sicherungstechnik zu ersetzen. So ist im Labor z.B. ein kompletter Führerstand samt eigenem Fahrzeugrechner vorhanden.

Abbildung 5: Führerstand als Teil des RailSiTe Labors

Das in dieser Arbeit erwähnte DSP - System soll die Ansteuerung eines solchen Fahrzeugrechners über eine Eurobalise ermöglichen. Zu diesem Zweck sollen Balisentelegramme, die vom RailSiTe - Labor an den DSP übertragen werden, kodiert und anschließend über eine Antenne an den Fahrzeugrechner gesendet werden können.

Für die Kommunikation zwischen RailSiTe - Labor und DSP steht eine RS 232 Schnittstelle zur Verfügung, für die ein Protokoll definiert wurde. In der vorhandenen DSP - Software stehen bereits Kommandos wie *„Balisentelegramm speichern“* und *„Sequenz von Balisentelegrammen senden“* zur Verfügung. Dieses Protokoll soll als Teil der Arbeit um weitere Kommandos, wie z.B. *„Balisentelegramme kodieren“* erweitert werden. Eine genaue Übersicht über das RS 232 Protokoll zur Kommunikation zwischen DSP und RailSiTe - Labor ist in [2] gegeben.

Über eine weitere RS 232 Schnittstelle können Diagnosemeldungen vom DSP an einen PC gesendet werden. Diese Übertragung findet bisher als Klartextmeldung unidirektional vom DSP - System zum PC statt. In einem weiteren Teil der Arbeit soll diese Diagnoseschnittstelle um ein einfaches Protokoll erweitert werden, dass es dem Benutzer ermöglicht, den DSP in einen Diagnose Modus zu versetzen, in dem Anforderungen des RailSiTe - Labors ignoriert und Diagnosevorgänge ausgeführt werden können. So soll es z.B. möglich sein, ein Balisentelegramm unendlich lange zyklisch zu senden. Die genauen Anforderungen an diese Schnittstelle sind in Kapitel 6.1 näher beschrieben.

Eine Übersicht über das DSP System als Bestandteil des RailSiTe - Labors ist in Abbildung 6 gegeben.

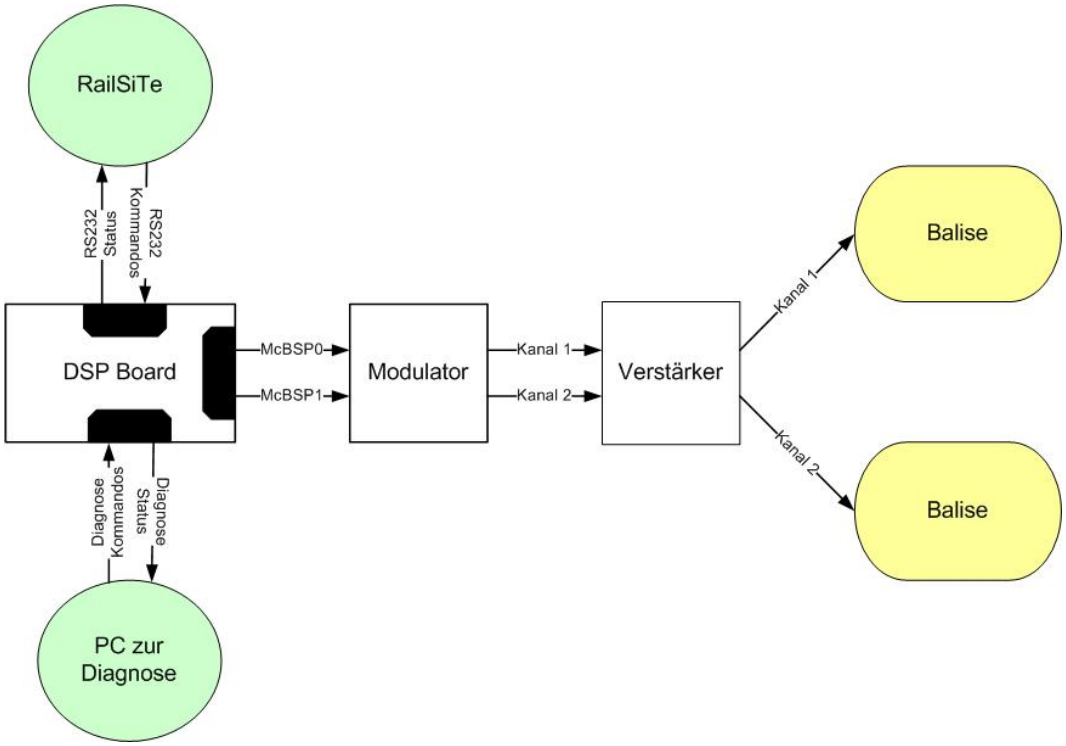

Abbildung 6: Übersicht DSP - System

In den nachfolgenden Kapiteln werden die Hardware des Zielsystems und die bereits implementierte Softwarestruktur beschrieben. In Kapitel 3.3 folgt schließlich eine grobe Beschreibung der Entwicklungsumgebung, die zur Durchführung der praktischen Programmieraufgaben verwendet wird.

3.2.1 Hardware

Als Hardware Plattform wird ein Microcontroller Evaluation Board der Firma Texas Instruments verwendet. Das DSK6713 Evaluation Board benutzt den Texas Instruments TMS320C6713 Microcontroller als Prozessor. Dabei handelt es sich um einem 32 Bit Digitalen Signal Prozessor (DSP), der mit einer Taktrate von 225 MHz betrieben wird. Er bietet bei dieser Taktrate einen Durchsatz von 1800 MIPS ($1800 \cdot 10^6$ Instruktionen pro Sekunde).

Der DSP bietet zahlreiche interne Peripheriebausteine, die auch in der DSP Software zur Balisenansteuerung verwendet werden. So enthält der DSP zwei McBSPs (engl. Multi Chanel Buffered Serial Port), über die digitale Signale als serieller Bitstrom gesendet und empfangen werden können. Diese McBSPs werden in der DSP Software als Ausgänge für die Balisentelegramme benutzt. Von dort werden die Datenströme über den Modulator und den nachfolgenden Verstärker an die Balise übertragen.

Der McBSP wiederum wird über einen EDMA Transfer (engl. Enhanced Direct Memory Access) mit den nötigen Telegrammdaten versorgt. Über diesen Transfer können Daten von einem Speicherbereich an einen anderen übertragen werden, ohne dass der Prozessor dadurch belastet wird.

Für zeitkritische Aufgaben stellt der DSP mehrere Timer / Counter zur Verfügung, über die z.B. das Timing beim Senden von Balisentelegrammen in einer Sequenz gesteuert wird. Eine genaue Beschreibung des DSP findet sich in [6]. Eine Übersicht über das Evaluation Board ist in Abbildung 7 gegeben.

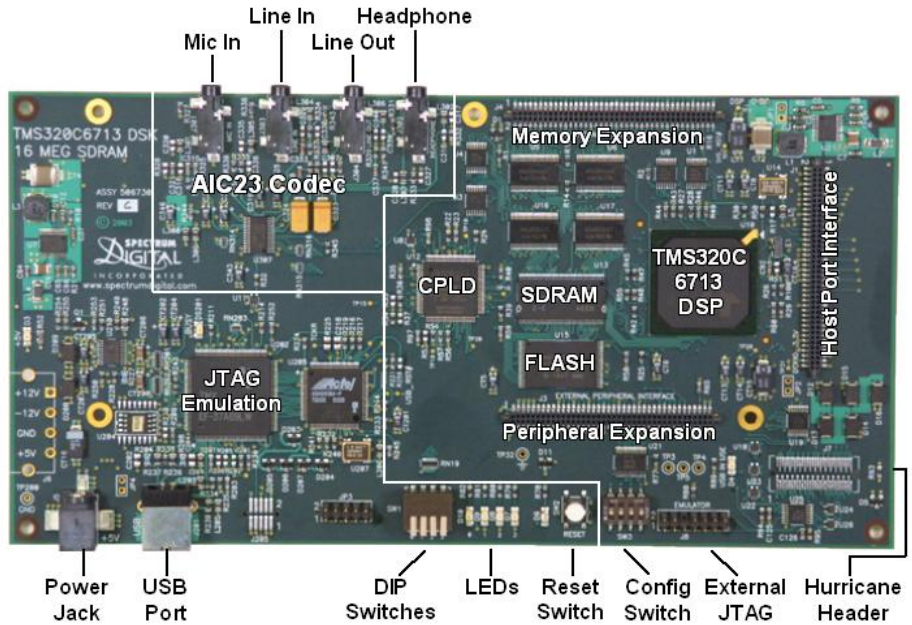

Abbildung 7: DSK6713 Evaluation Board (Quelle: DSK6713 Online Hilfe)

Neben dem DSP enthält das Board diverse Schnittstellen für Peripherie, wie z.B. einen AIC 23 tauglichen Audio Codec. Außerdem stellt das Board einen 16 MByte großen externen Speicher (SDRAM) zur Verfügung, in dem auch große Datenstrukturen problemlos abgelegt werden können.

Zur Verbindung mit der Entwicklungsumgebung steht ein JTag Interface zu Verfügung, dass über einen USB 2.0 Port mit einem PC verbunden werden kann. Über diese Schnittstelle ist es möglich, den Prozessor während der Laufzeit zu stoppen und so online zu debuggen.

Untypischerweise bietet weder der DSP noch das DSK6713 Evaluation Board eine RS 232 Schnittstelle zur Kommunikation mir anderen Geräten. Im DSP besteht zwar die Möglichkeit, eine RS 232 Schnittstelle mit Hilfe der McBSPs zu emulieren, diese werden allerdings in der DSP - Software bereits zur Balisenansteuerung benutzt und stehen deshalb nicht zu Verfügung.

Aus diesem Grund wird das Board um eine Daughtercard (engl. Tochterkarte, meint hier Erweiterungskarte) erweitert, die das Board um mehrere RS 232 Schnittstellen ergänzt. Diese UART (engl. Universal Asynchronous Receiver Transmitter) Erweiterungskarte (siehe Abbildung 8) der Firma DSP Global besitzt in der im Aufbau verwendeten Version vier zusätzliche RS 232 Schnittstellen, welche die Verbindungen zum RailSiTe Labor und zur Diagnose bieten. Sie wird auf das Erweiterungsinterface des DSK6713 aufgesteckt und so in den Speicherbereich des DSP eingeblendet.

Eine genaue Beschreibung der Erweiterungskarte ist in [7] gegeben.

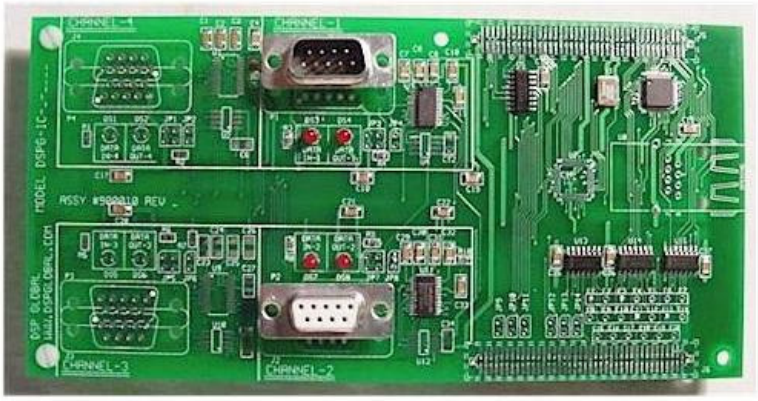

Abbildung 8: RS 232 Daughtercard (Quelle: www.dspglobal.com)

Die eigentlichen Digitaldaten der Balisentelegramme, die über die McBSPs an den Modulator gesendet werden, werden dort in analoge HF Signale umgesetzt, die später über die Antenne gesendet werden. Bei diesem Modulator handelt es sich um einen CMOS Umschalter. Dieser Umschalter macht eine harte FSK zwischen den beiden Frequenzen 3,951 MHz (low Pegel) und 4,516 MHz (high Pegel).

Die beiden Signale werden dabei durch zwei Signalgeneratoren der Firma HP (auch Agilent) zur Verfügung gestellt. Ein weiteres Signal aus dem DSP sorgt dafür, dass am Ausgang des Modulators 0 V ausgegeben werden, wenn kein Balisentelegramm gesendet wird. Dazu wird ein General Purpose I/O (engl. meint hier allgemein zu verwendender Ausgang) des DSP nur dann gesetzt, wenn über den McBSP ein Telegramm ausgegeben wird. Das resultierende HF Signal am Modulatorausgang wird mit diesem Signal UND verknüpft.

In einem nachfolgenden Verstärker wird das Signal soweit aufbereitet, dass es über die Antenne gesendet werden kann. Eine Übersicht über diesen Aufbau ist für einen Balisenkanal exemplarisch in Abbildung 9 gegeben.

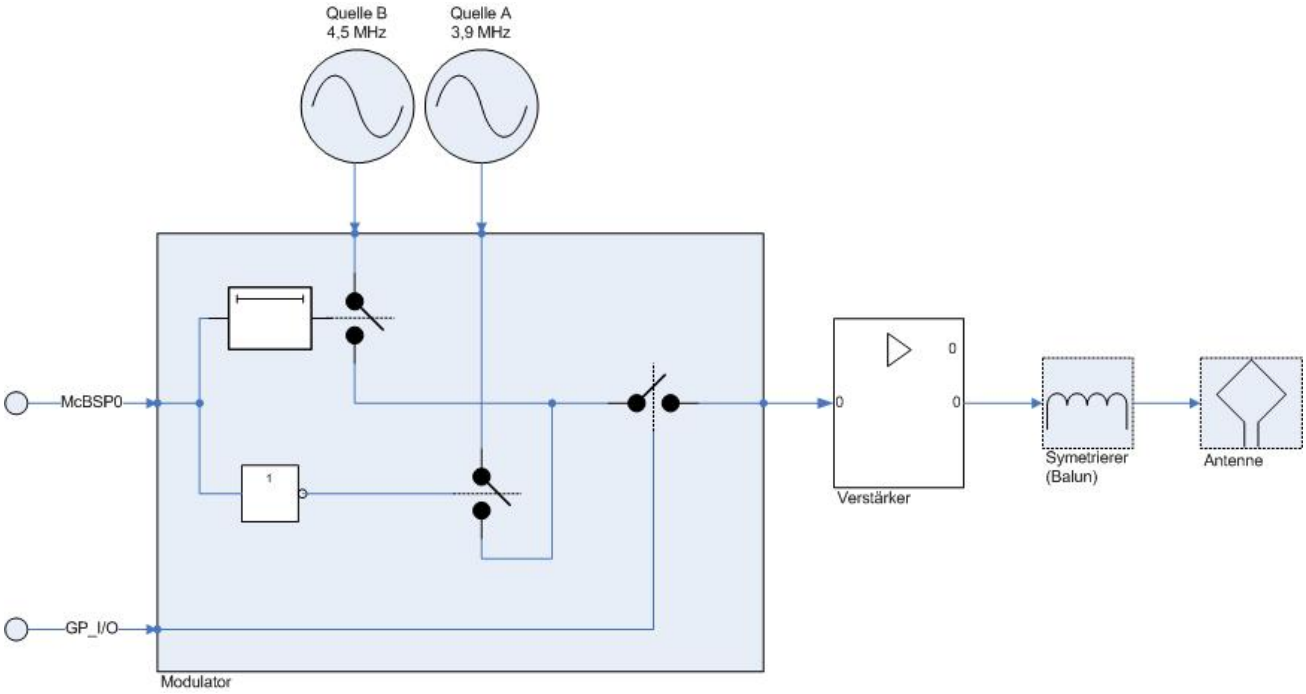

Abbildung 9: Übersicht Modulator

Dieser Modulator erzeugt durch die harte Frequenzumtastung Phasensprünge im Ausgangssignal. Diese können im Empfänger dazu führen, dass die Datensignale nicht fehlerfrei empfangen werden können. Aus diesem Grund ist es Teil dieser Arbeit, einen neuen Modulator aufzubauen, der am Ausgang einen stetigen Phasenverlauf besitzt. Der Aufbau dieses Modulators ist in Kapitel 7 gegeben.

3.2.2 Software

Die vorhandene DSP - Software bietet bereits die Möglichkeiten, Telegramme im DSP zu speichern, und diese innerhalb einer Sequenz über den HF - Kanal (Balise) zu versenden. Dazu ist die Firmware in folgende Teilbereiche gegliedert, die jeweils in einer eigenen Datei implementiert sind:

- **datahandler.c**

  In dieser Datei sind alle Funktionen implementiert, die für die Handhabung der Balisentelegramme notwendig sind. Sie enthält vor allem Funktionen, die für das Aufbereiten der Balisentelegramme vor dem Versenden notwendig sind.

- **protocol.c**

  Diese Datei enthält alle Funktionen, die für das RS 232 Protokoll zum RailSiTe Labor notwendig sind. Es enthält neben allen Unterfunktionen für die Kommandos des RailSiTe - Protokolls auch die Hauptschleife (**mainLoop (…)**) in der sich der DSP während der gesamten Programmausführung befindet.

- **mcbsp.c**

  In dieser Datei sind alle Funktionen implementiert, die zu Handhabung des EDMA Kanals und der McBSPs notwendig sind. Vor allem sind hier die Funktionen enthalten, die den EDMA Kanal vor dem Versenden der Telegramme konfigurieren, damit er nach dem Start des EDMA Kanals selbsttätig die Telegrammdaten an den McBSP sendet.

- **timer2antennas.c**

  Diese Datei enthält die eigentliche ISR, die das Versenden von Telegrammen innerhalb einer Sequenz steuert. Außerdem sind hier alle Funktionen enthalten, die den DSP für den zeitlichen Ablauf des Sendens einer Sequenz vorbereiten.

- **uart.c**

  Diese Datei enthält neben der ISR für den UART auch alle Funktionen, mit denen direkt auf den Puffer der UART Schnittstellen zugegriffen werden kann.

- **irq.c**

  In dieser Datei ist eine Funktion zur Initialisierung aller benötigten Interrupts vorhanden.

Nach dem Start des DSP (Reset oder Power On) wird die Funktion **main (…)** aufgerufen. Sie stellt den Einsprungpunkt für die DSP - Software nach einem Reset dar. In der Funktion **main (…)** wird zuerst die Funktion **initComponents (…)** aufgerufen, die alle Variablen, Parameter und Peripheriekomponenten initialisiert.

Nach einer erfolgreichen Ausführung dieser Funktion verzweigt der DSP in die Funktion **mainLoop (…)**. In dieser Funktion wird auf Pakete des RailSiTe - Labors gewartet. Nach dem Empfang eines RailSiTe - Pakets wird die zugehörige Unterfunktion aufgerufen, um das

RailSiTe - Kommando zu bearbeiten. Die Funktion **mainLoop (…)** wird nach Aufruf nie wieder verlassen.

Eine Ausnahme bieten die Interrupt Service Routinen (ISR), die durch den DSP aufgerufen werden, wenn ein Ereignis an einer der Peripheriekomponenten auftritt. In der Firmware sind bereits zwei ISRs implementiert.

- **ext_int (…)**

  Diese ISR wird vom DSP aufgerufen, wenn ein externer Interrupt durch den UART auf der Daughtercard ausgelöst wurde. Er handhabt das Versenden und Empfangen von Daten über die beiden RS 232 Schnittstellen und arbeitet im DSP auf jeweils einem Ringpuffer für Senden und Empfangen pro Schnittstelle. Die Ringpuffer arbeiten dabei nach dem First In First Out (FIFO) Prinzip.

- **timerISR (…)**

  In dieser ISR wird das Senden von Telegrammen innerhalb einer Sequenz über die HF - Schnittstelle gesteuert. Sie wird durch den Timer 1 ausgelöst. Dazu wird die Periode des Timer 1 immer so eingestellt, dass das nächste Ereignis des Timer 1 genau zum Zeitpunkt des nächsten Ereignisses innerhalb einer Sequenz auftritt. Die Erstellung des zeitlichen Ablaufs aus den Sequenzdaten übernimmt dabei die Funktion **doSequenceTiming (…)**.

Wenn das RailSiTe - Labor eine Sequenz über die HF - Schnittstelle (Balise) senden möchte, so muss sie zuerst alle in der Sequenz benötigten Balisentelegramme an den DSP senden. Anschließend schickt das RailSiTe das Kommando *„RS2DSP_DefAndRunSeqc“*. Dieses Paket enthält alle zeitlichen Parameter, die für den Start einer Sequenz im DSP notwendig sind.

Zu seiner Bearbeitung wird im DSP aus der Funktion **mainLoop (…)** in die zugehörige Unterfunktion **ID_handle_RS2DSP_DefAndRunSeqc (…)** verzweigt. Dort werden die Telegramme zum versenden vorbereitet und anschließend die Funktion **doSequenceTiming (…)** aufgerufen, die den zeitlichen Ablauf der Sequenz vorbereitet. Am Ende von **doSequenceTiming (…)** wird der Timer 1 für das erste Ereignis der Sequenz vorbereitet und gestartet.

Die restliche Abarbeitung der Sequenz findet dann ausschließlich in der Funktion **timerISR (…)** statt. Dort wird nach der Behandlung des aktuellen Ereignisses der Timer 1 für das nächste Ereignis bereit gemacht. Dabei folgt in jedem Kanal auf eine Sendeperiode immer eine Pauseperiode. Wenn alle Ereignisse einer Sequenz abgearbeitet sind, wird der Timer 1 wieder gestoppt. Diesen Vorgang beschreibt Abbildung 10.

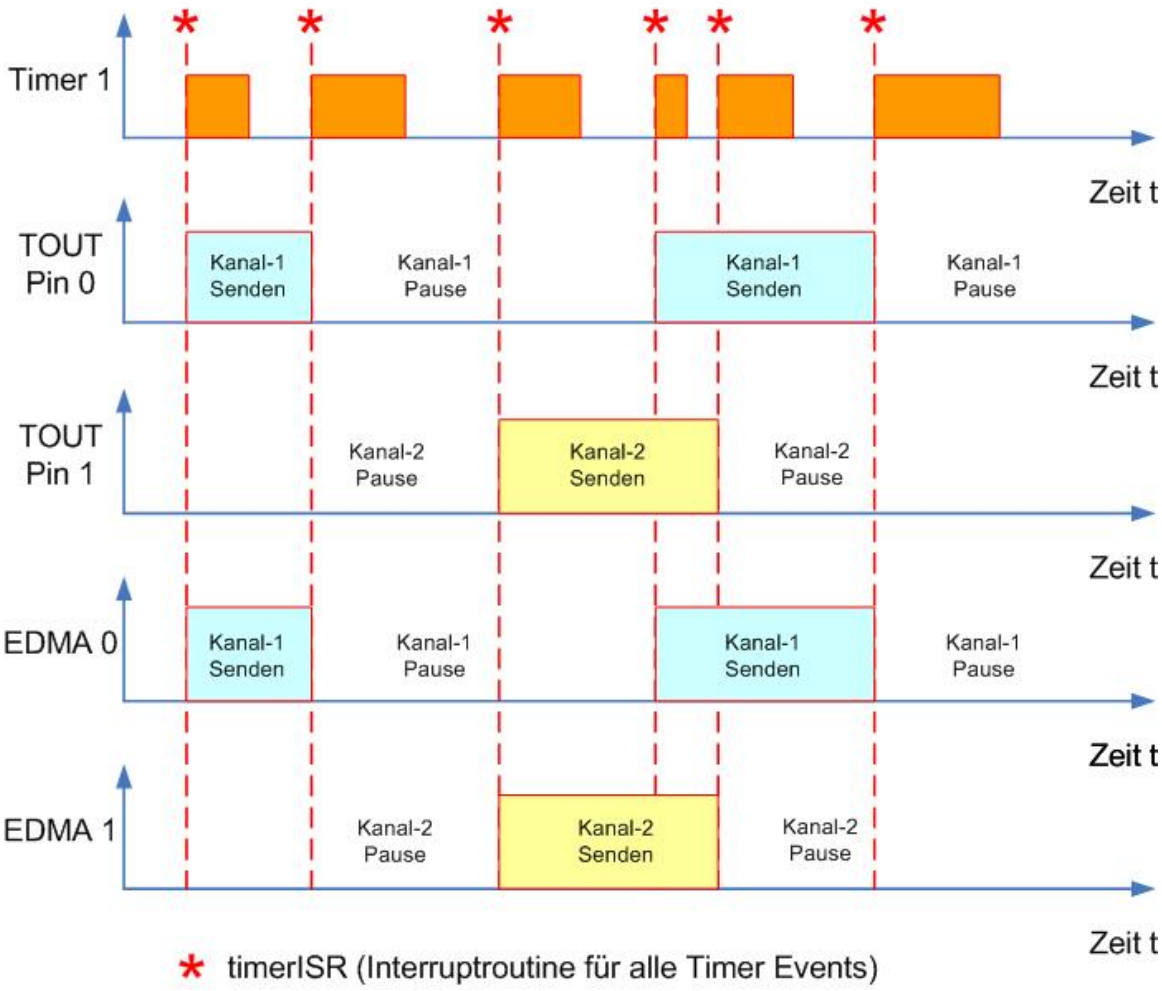

Abbildung 10: Übersicht Ablauf bei Senden einer Sequenz

3.3 Zu verwendende Tools

Als Entwicklungsumgebung wird das Code Composer Studio (CCS) der Firma Texas Instruments in der Version 2.0 verwendet. Dabei handelt es sich um eine komplette integrierte Entwicklungsumgebung (engl. Integrated Development Envirement, kurz IDE).

Neben allgemeinen Funktionen zum Quellcode und Projektmanagement bietet es einige spezielle Werkzeuge für die Entwicklung. Die wichtigsten dieser Werkzeuge werden nachfolgend näher erläutert.

- **Debugger**

  Der Debugger des Code Composer Studio bietet die Möglichkeit, den Prozessor zu jeder Zeit während des Betriebs anzuhalten, oder den Code in Einzelschritten auszuführen. Zusätzlich können die Inhalte aller Variablen und Register in einem Überwachungsfenster (engl. Watchwindow) analysiert werden.

- **Profiler**

  Mit dem Profiler bietet das CCS die Möglichkeit, die Dauer einzelner Funktionen und Codeabschnitte zu messen. Das ist insbesondere bei der Entwicklung von Firmware für Microcontroller wichtig, da hier meist Echtzeitanforderungen an das System gestellt werden.

- **Optmizer**

  Mittels des Optimizers (engl. Optimierer) kann der Quellcode automatisch für die Abarbeitung auf dem Zielsystem (TSM320C6713) optimiert werden. Dazu stellt das CCS fünf Stufen der Optimierung zur Verfügung. In den beiden höchsten Stufen wird die prozessorinterne Softwarepipeline zu parallelen Ausführung von Befehlen aktiviert. Der Quellcode wird dann, besonders in Schleifen, auf die parallele Ausführung optimiert. Dieses Vorgehen kann allerdings Probleme bei manchen Programmstrukturen erzeugen, die nach der Optimierung möglicherweise nicht mehr korrekt ausführbar sind. Im Kapitel 4.3.4 wird genauer auf die Eigenschaften und die Bedienung des Optimizers eingegangen.

- **DSP / BIOS Konfigurations-Tool**

  Über das DSP / BIOS Konfigurationstool ist es möglich, alle Peripheriekomponenten des Microcontrollers über ein graphisches Menü zu konfigurieren. Alle Konfigurationsdaten werden später automatisch in Kode umgesetzt, der dann für die Kompilierung und das Linken der Firmware benutzt wird.

Ein Nachteil des CCS in der hier verwendeten Version ist, dass das CCS ein angeschlossenes DSP - Board als Bedingung voraussetzt, um zu starten. Dadurch kann nur am Quellcode gear-

beitet werden, wenn ein DSP - Board mit dem PC verbunden ist. Eine Übersicht über die Oberfläche des CCS zeigt Abbildung 11.

Als zweites Tool wurde das Versionsverwaltungssystem Subversion benutzt. Dabei handelt es sich um ein Freeware Tool, das den Quellkode auf einem zentralen Server verwaltet. Jeder Mitarbeiter kann sich eine Arbeitskopie (engl. Working Copy) des Quellcodes auf seinen eigenen PC laden und dort bearbeiten.

Hat er seine Bearbeitung abgeschlossen, muss er die aktualisierten Quelldateien wieder an den Server zurückschicken, um sie allen Nutzern zugänglich zu machen. Subversion ist dabei nicht restriktiv. Das heißt, dass mehrere Anwender zur gleichen Zeit an denselben Dateien arbeiten können. Sollte dies der Fall sein, so meldet Subversion mögliche Konflikte beim Hochladen der Dateien auf den Server an alle betroffenen Benutzer.

Diese sind dann selbst dafür verantwortlich, alle Änderungen zu einer gemeinsamen neuen Datei zu vereinen.

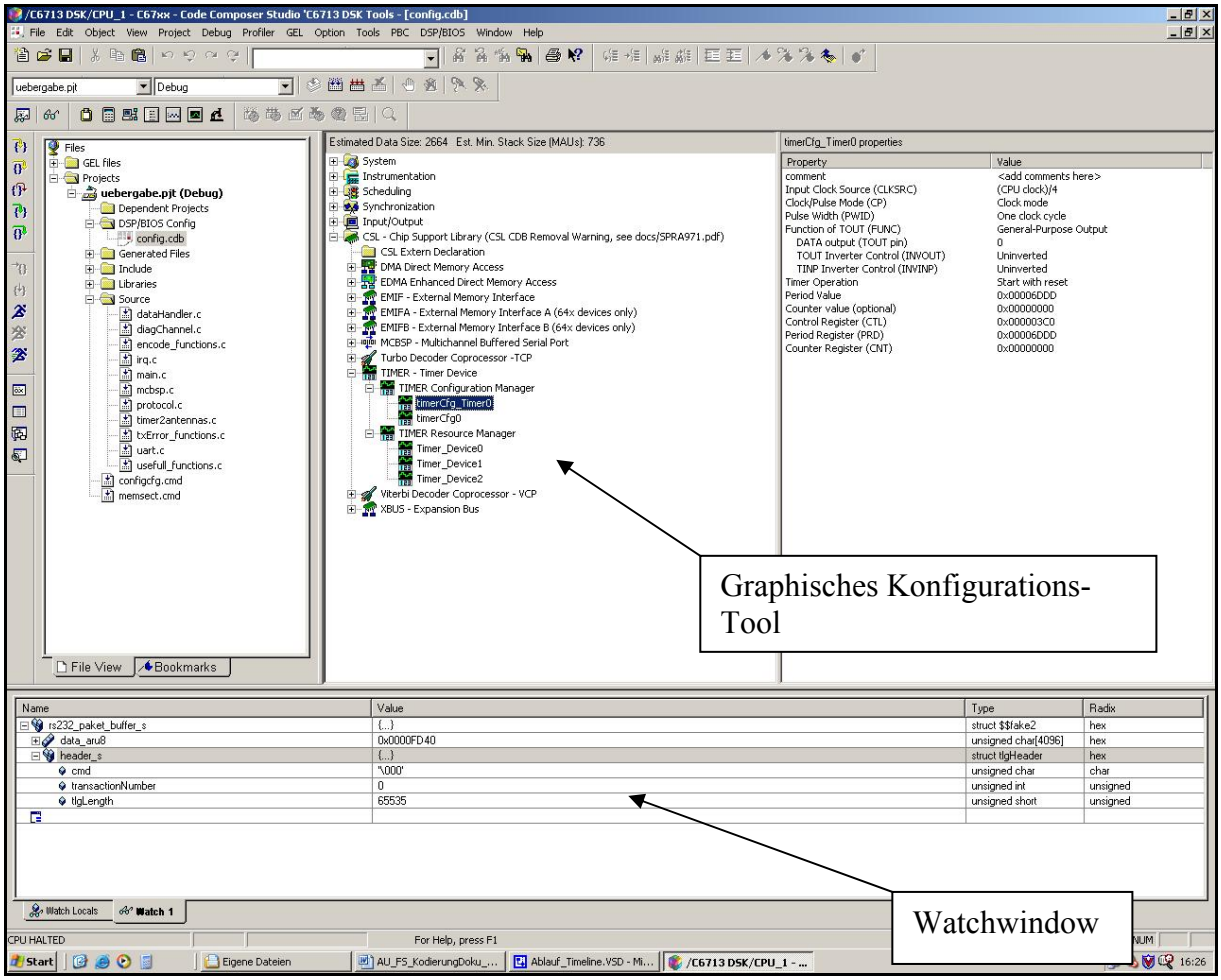

Abbildung 11: Übersicht Code Composer Studio (Version 2.0)

4 Kodierung der Telegramme

4.1 Einführung

Das ETCS System bietet über die Eurobalisen die Möglichkeit, Daten von der Strecke über eine Funkschnittstelle an einen Fahrzeugrechner zu übertragen. Dazu werden von den Eurobalisen Datentelegramme übertragen, wenn sie von einem Fahrzeug mit ETCS Fahrzeugrechner überfahren werden. Diese Art der punktuellen Datenübertragung gleicht anderen PZB Systemen, wie z.B. Indusi. Allerdings bietet ETCS hier durch die Übertragung von Datentelegrammen mit einer Datenrate von 565 kBit/s und Überfahrtgeschwindigkeiten von bis zu 500 km/h starke Leistungssteigerungen gegenüber Indusi und PZB 90, bei denen praktisch nur wenige Zustände überwacht werden können (offener / geschlossener Gleismagnet…).

Durch die hohe Datenrate können für den Bahnverkehr vergleichsweise hohe Datenmengen an den Zug übertragen werden. So stellt ETCS für die Übertragung über die Eurobalisen zwei Telegrammtypen zur Verfügung: Ein kurzes, und ein langes Telegramm.

Die beiden Telegrammtypen unterscheiden sich nur in der Menge der Nutzdatenbits. Dort stehen in einem kurzen Telegramm 210 Bit und in einem langen Telegramm 830 Bit zur Verfügung. Beiden Telegrammtypen sind zusätzliche Bits zur Sicherung gegen Übertragungsfehler und die Durchführung der Kanalkodierung der Telegramme angehängt. Der grundsätzliche Aufbau eines Balisentelegramms unter ETCS ist nachfolgend aufgezeigt.

Tabelle 2: Aufbau eines Balisentelegramms

Shaped Bits	**Cb**	**Sb**	**Esb**	**CheckBits**
(kodierte Nutzdaten)	(Kontrollbits)	(Scramblingbits)	(Extra Shapingbits)	(Checksumme)
913 Bit (long)	3 Bit	12 Bit	10 Bit	85 Bit
231 Bit (short)				

Um die Nutzdaten der Balisentelegramme möglichst fehlerfrei über die Funkschnittstelle übertragen zu können, werden sie vor der Übertragung durch einen komplizierten Algorhythmus kodiert und mit zusätzlicher Redundanz (zusätzlichen Bits) versehen. Auf diese Weise ergibt sich eine Länge von 341 Bit für ein kurzes und 1023 Bit für ein langes Telegramm. Ein langes Balisentelegramm ist also genau dreimal so lang, wie ein kurzes Telegramm.

Da die Balisentelegramme bei überfahren einer Balise zyklisch wiederholt werden, muss die Kanalkodierung der Telegramme neben dem Schutz gegen allgemeine Übertragungsfehler auch einen Schutz gegen Verwechslung von kurzen und langen Telegrammen gewährleisten. Um all diese Anforderungen zu erfüllen, ist die Kodierung der Balisentelegramme in drei Schritte unterteilt. Diese sind:

1. Verwürfelung der Nutzdaten (mit Hilfe der Scramblingbits)
2. Substitution der verwürfelten Nutzdaten durch Kodeworte eines Kodes mit höherer Hammingdistanz
3. Berechnung einer Checksumme

Im ersten Schritt werden die Nutzdaten mit Hilfe eines Schieberegisters verwürfelt. Diese Verwürfelung soll später einen besseren Schutz gegen Burstfehler bieten, die beim Dekodieren der Telegramme in Einzelbitfehler umgewandelt werden. Dabei dienen die 12 Scramblingbits (SB) im Telegramm als Initialwerte für das Schieberegister, mit dem die Nutzdaten verwürfelt werden. Am Ende dieses Schrittes besteht das Telegramm aus 10 Bit langen Datenworten, welche die verwürfelten Nutzdaten enthalten.

Im zweiten Schritt werden diese Datenworte durch einen anderen Kode substituiert. Dabei wird pro 10 Bit reine Nutzdaten ein Bit Redundanz im neuen Kode hinzugefügt. Am Ende besteht das Telegramm also aus 11 Bit breiten Datenworten. Die einzelnen Kodeworte des Substitutionskodes wurden so ausgelegt, dass sie eine möglichst große Hammingdistanz zueinander aufweisen.

Im letzten Schritt wird eine 85 Bit lange Checksumme über das gesamte Telegramm von den Shaped Bits bis einschließlich der Extra Shaping Bits berechnet und an das fertig kodierte Telegramm angehängt. Hier ist zu beachten, dass die Checksumme, genau wie Cb, Sb und Esb auch, nur aus Kodeworten bestehen darf, die im Substitutionskode vorhanden sind. Dadurch besteht das fertig kodierte Telegramm nur noch aus gültigen Kodeworten des Substitutionskodes.

Da bei der Berechnung des Checksumme nicht sichergestellt werden kann, dass die erzeugte Checksumme nur aus gültigen Kodeworten besteht, muss sie unter Umständen so lange neu berechnet werden, bis diese Bedingung erfüllt ist. Um die Nutzdaten nicht ändern zu müssen, werden dem Telegramm hierzu 10 Extra Shaping Bits (ESB) hinzugefügt, die in jedem Durchlauf der Berechnung der Checksumme geändert werden, bis die Checksumme nur noch aus gültigen Kodeworten besteht. Sollte es nicht möglich sein eine gültige Checksumme unter Verwendung aller 2^{10} ESB - Kombinationen zu erzeugen, so muss die Kodierung im Schritt 1 unter Verwendung anderer Scrambling Bits erneut begonnen werden.

Am Ende der Kodierung wird das Telegramm auf bestimmte Bedingungen (engl. Conditions / Constraints) überprüft. Diese sind:

- **„Alphabet Condition“**

  Diese Bedingung überprüft, ob das fertig kodierte Telegramm nur aus gültigen Kodeworten des Substitutionskodes besteht.

- **„Off-Sync-Parsing Condition“**

  Diese Bedingung ist notwendig, da die Balisentelegramme zyklisch gesendet werden. Der Empfänger kann also an einer beliebigen Stelle innerhalb eines Telegramms mit dem Empfang beginnen. Um zu sichern, dass trotzdem die richtigen Kodeworte gelesen werden, überprüft diese Bedingung, ob auch dann Folgen von gültigen Kodeworten entstehen, wenn nicht am Anfang des Telegramms mit dem Empfang begonnen wird.

⇨ **„Aperiodicity Condition“**

Diese Bedingung ist notwendig, da die Balisentelegramme zyklisch gesendet werden. Sie überprüft ausschließlich lange Telegramme und stellt sicher, dass sich diese innerhalb des Telegramms nicht periodisch wiederholen. Ansonsten könnten sie im Empfänger mit kurzen Telegrammen verwechselt werden.

⇨ **„Undersampling Condition“**

Diese Bedingung stellt sicher, dass bei Über- und Unterabtastung der Telegramme keine gültigen Kodeworte entstehen können.

Wie aus diesen Bedingungen ersichtlich ist, ist die Kodierung eines Balisentelegramms ein iterativer Prozess. Bei der Kodierung kann weder eine definierte Kodierdauer angegeben, noch überhaupt versichert werden, dass ein Telegramm kodierbar ist. Die Wahrscheinlichkeit, eine gültige Kombination von SB und ESB für ein Nutzdatentelegramm zu finden ist jedoch sehr hoch, da für jedes Telegramm eine Menge von

$$N = 2^{LEN\_SB} \cdot 2^{LEN\_ESB} = 2^{12} \cdot 2^{10} = 2^{22}$$

mögliche Kodierungen zur Verfügung stehen. Sollte keine der SB/ESB Kombinationen zu einem gültigen Telegramm führen, so müssen die enthaltenen Nutzdaten geringfügig geändert werden. Die komplette Spezifikation der Kodierung ist in [1] enthalten.

Die nachfolgenden Kapitel beschreiben die Implementation des Kodierungsalgorhythmus in die DSP - Software. Am Ende des Kapitel 4 wird eine Performanceuntersuchung des implementierten Algorhythmus gemacht, um mögliche Einschränkungen für die Verwendung im RailSiTe - Betrieb feststellen zu können.

4.2 Umsetzung der Kodierung auf dem DSP

Die Kodierung der Telegramme wird im DSP durch das RS232 Kommando **RS2DSP_EncodeTlg** (siehe [2], Seite 19) angestoßen, dass vom RailSiTe - Labor an den DSP gesendet wird. Als Eingangswerte werden vom BTM des RailSiTe - Labors alle Informationen übertragen, die für die Kodierung notwendig sind. Diese sind:

Tabelle 3: Parameter des Kommandos RS2DSP_EncodeTlg

Name	Typ	Beschreibung
nSeq	uint32	Nummer des Telegramms, das codiert werden soll.
sbInit	uint16	Startwerte für die Scrambling Bits bei der Kodierung der Telegramme. Der Wert 0xFFFF generiert für jedes Telegramm einen zufälligen Wert für die Scramblingbits.
esbInit	uint16	Startwerte für die Extra Shaping Bits bei der Kodierung der Telegramme. Der Wert 0xFFFF generiert für jedes Telegramm einen zufälligen Wert für die Extra Shaping Bits.

Die zu kodierenden Daten wurden schon vorher durch einen **RS2DSP_StoreTlg** (siehe [2], Seite 13) Befehl vom BTM in einem Telegrammspeicherplatz des DSP hinterlegt. Dadurch kann die Kodierung nach dem Empfang des **RS2DSP_EncodeTlg** sofort beginnen.

Dazu wird im Handler für das RailSiTe - Protokoll im DSP bei Empfang des Kommandos **RS2DSP_EncodeTlg** die Funktion **handle_RS2DSP_EncodeTlg (…)** aufgerufen, die die Kodierung im DSP ausführen soll.

Diese Funktion gehört zum allgemeinen RS 232 Protokoll, das zur Kommunikation zwischen dem RailSiTe - Labor und dem DSP - Board genutzt wird und ist in der Datei **protocol.c** definiert. In dieser Funktion wird der eigentliche Kodiervorgang mit Aufruf der Funktion **encodeTelegram (…)** gestartet. Diese Funktion ist wiederum in drei große Blöcke unterteilt. Diese sind:

srcambleAndSubst (…):

Diese Funktion führt die eigentliche Verschlüsselung der Benutzerdaten durch. Anschließend werden die verschlüsselten Daten über eine Umsetzungstabelle in gültige Kodeworte des Substitutionskodes umgesetzt.

polyRemainder (…):

Diese Funktion berechnet die Checksumme über alle bereits verschlüsselten Datenbits.

checkConstraints (…):

Diese Funktion überprüft alle Bedingungen, die das fertig kodierte Telegramm erfüllen muss, damit es als gültiges Telegramm über die Balise an den Zug übertragen werden darf.

Die Funktion **encodeTelegram(…)** und die in ihr aufgerufenen Funktionen wurden zum großen Teil aus der Software zum BTM/i übernommen und an die DSP - Software angepasst. Sie werden alle in der Datei **encode_functions.c** definiert. Dieses Vorgehen bietet den Vorteil, dass die bereits hoch optimierte Software, bis auf nötige Anpassungen, nicht erneut programmiert werden muss. Eine Übersicht über den Ablauf der Funktion **handle_RS2DSP_EncodeTlg** ist in Abbildung 12 gegeben:

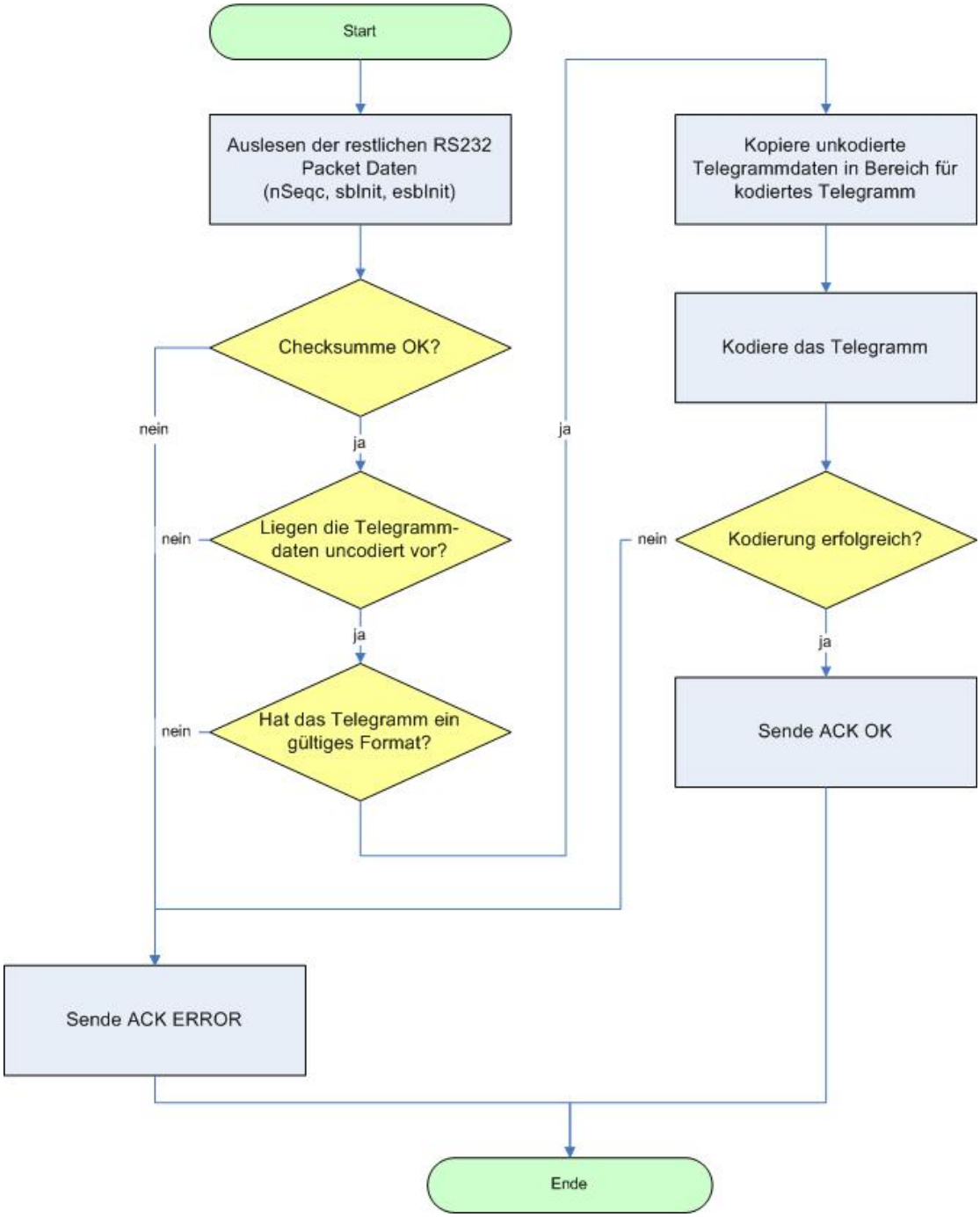

Abbildung 12: Flussdiagramm handle_RS2DSP_EncodeTlg (…)

In den nachfolgenden Kapiteln sollen alle Unterfunktionen genauer erläutert werden.

4.2.1 Funktion encodeTelegram (…)

Die Funktion **encodeTelegram (…)** dient der Kodierung der Telegramme über den in der Norm spezifizierten Algorhythmus. Sie bekommt als Parameter alle Eingangsgrößen übergeben, die für die Kodierung notwendig sind und gibt die fertig kodierten Daten in einem uint8 Feld als Bitfeld zurück. Im Folgenden werden die Parameter der Funktion genauer erläutert:

Tabelle 4: Parameter der Funktion encodeTelegram (…)

Name	Typ	Beschreibung
tlg	uint8*	Zeiger auf das Bitfeld der unkodierten Benutzerdaten
def_sb	uint16*	Zeiger auf das Wort mit den 12 Scramblebits, die für die Kodierung verwendet werden sollen
def_esb	uint16*	Zeiger auf die 10 Extra Shaping Bits, die für die Kodierung verwendet werden sollen
tlg_format	uint8	Typ des zu kodierenden Telegramms TLG_TYPE_SHORT: kurzes Telegramm TLG_TYPE_LONG: langes Telegramm

Tabelle 5: Rückgabewert der Funktion encodeTelegram (…)

Name	Typ	Beschreibung
-	-	-

Zusätzlich wird ein Array **substitutionWords[1024]** mit allen gültigen 11 Bit Datenworten angelegt. Er dient später als Substitutionstabelle für die 10 Bit Datenworte. Ein zweiter Array **validWords[2048]** enthält für jedes der möglichen 2048 11 Bit Datenworte einen boolean Eintrag. Dieser ist *True* für alle gültigen Datenworte und ansonsten *False*. Auf diese Wiese kann später in der Funktion **checkConstaints (…)** einfach festgestellt werden, ob ein Datenwort gültig ist.

Da die Telegrammdaten zu Beginn der Kodierung noch als Bitfeld im Speicher des DSP vorliegen, werden sie zu Beginn der Funktion **encodeTelegram (…)** in einen Array **U[83]** aus 10 Bit Worten umgewandelt. Da der Prozessor keine 10 Bit großen Variablen unterstützt, wird für jedes 10 Bit Wort, und auch später für die 11 Bit breiten Worte, eine 16 Bit Variable verwendet.

Die Größe des Array ist dabei immer auf ein langes Telegramm zugeschnitten. Sie berechnet sich wie folgt:

$$< Größe\_des\_Array > k = \frac{< Anzahl\_Nutzbits\_langes\_T\lg. > m = 830}{< Anzahl\_Bit\_pro\_Wort > n = 10} \qquad (1)$$

Formel 1: Berechnung der Arraygröße für U[]

Er besitzt also immer 83 Worte (830 Bits / 10 Bit pro Wort = 83 Worte). Allerdings werden bei der Kodierung von kurzen Telegrammen nur die ersten 21 Einträge verwendet (210 Bits / 10 Bit pro Wort = 21 Worte).

Nach der Umwandlung der Rohdaten in den Array **U[]** wird ein zweiter Array **US[83]** angelegt, der bis auf das höchste Datenwort **US[k-1]** mit **U[]** identisch ist. Das höchste Datenwort **U[k-1]** wird durch die Summe aller Datenworte ersetzt. Diese Substitution soll erreichen, dass sich selbst bei Veränderung nur weniger Bits der Nutzdaten ein völlig neues Telegramm ergibt. Das neue Wort ergibt sich aus folgender Formel:

$$US[k-1] = \sum_{i=0}^{k-1} U[i] \bmod 2^{10} \qquad \textbf{(2)}$$

Formel 2: Berechnung eines neuen Wortes US[k-1]

Im nächsten Schritt sollen aus den 10 Bit breiten unkodierten Daten im Array **US[]** die fertig kodierten 11 Bit breiten Datenworte erzeugt werden. Dazu wird die Funktion **scrambleAndSubst (…)** aufgerufen.

Sie verwürfelt die Daten und gibt das Ergebnis als fertig verschlüsselte 11 Bit Datenworte im Array **US[]** zurück. Der vorherige Inhalt des Arrays wird dabei überschrieben. Das Ergebnis der Verwürfelung wird mit der Funktion **checkConstraints (…)** auf die Erfüllung aller Bedingungen überprüft, welche die Norm an ein fertig kodiertes Balisentelegramm stellt. Dabei wird an diesem Punkt keine Überprüfung der Alphabet Bedingung durchgeführt, da alle Datenworte durch die Substitution gültig sein müssen. Sollte eine der Bedingungen verletzt werden, so werden die Nutzdaten mit anderen Scramblingbits erneut verwürfelt.

Erst wenn der Nutzdatenteil fertig verwürfelt ist und alle Bedingungen erfüllt sind, wird in einer zweiten Schleife die Checksumme berechnet und an das Telegramm angehängt. Hierzu wird die Funktion **polyRemeainder (…)** aufgerufen.

Sie führt die Berechnung der Checksumme über den gesamten Nutzdatenteil plus Scramblingbits, Extra Shaping Bits und Control Bits durch. Dabei haben die Extra Shaping Bits eine besondere Funktion. Da auch die Checksumme der Alphabet Bedingung gehorchen muss, kann der Nutzdatenteil über die Extra Shaping Bits selbst nach der Verwürfelung geändert werden. Dadurch ergibt sich für jede Extra Shaping Bit Kombination bei gleich bleibenden Nutzdaten und Scramblingbits eine neue Checksumme.

So wird die Checksumme so oft neu berechnet, bis sie der Alphabet Bedingung entspricht, oder alle 2^{10} Extra Shaping Bit Kombinationen durchprobiert wurden. Für den Fall, dass keine der Extra Shaping Bit Kombinationen zu einer gültigen Checksumme führt, wird die Verwürfelung mit neuen Scramblingbits gestartet.

Dieses Vorgehen sorgt dafür, dass das Kodieren eines Telegramms unter Umständen sehr lange dauern kann. Deshalb muss seitens des RaiSiTe - Labors dafür gesorgt werden, dass vor dem Versenden eines Telegramms in einer Sequenz genug Zeit für die Ausführung des Kodierbefehls zur Verfügung steht. Genaue Messungen und Performance Untersuchungen des Kodierungsalgorhythmus folgen in Kapitel 4.3.

Eine Übersicht über die Funktion **encodeTelegram (…)** ist nachfolgend gegeben.

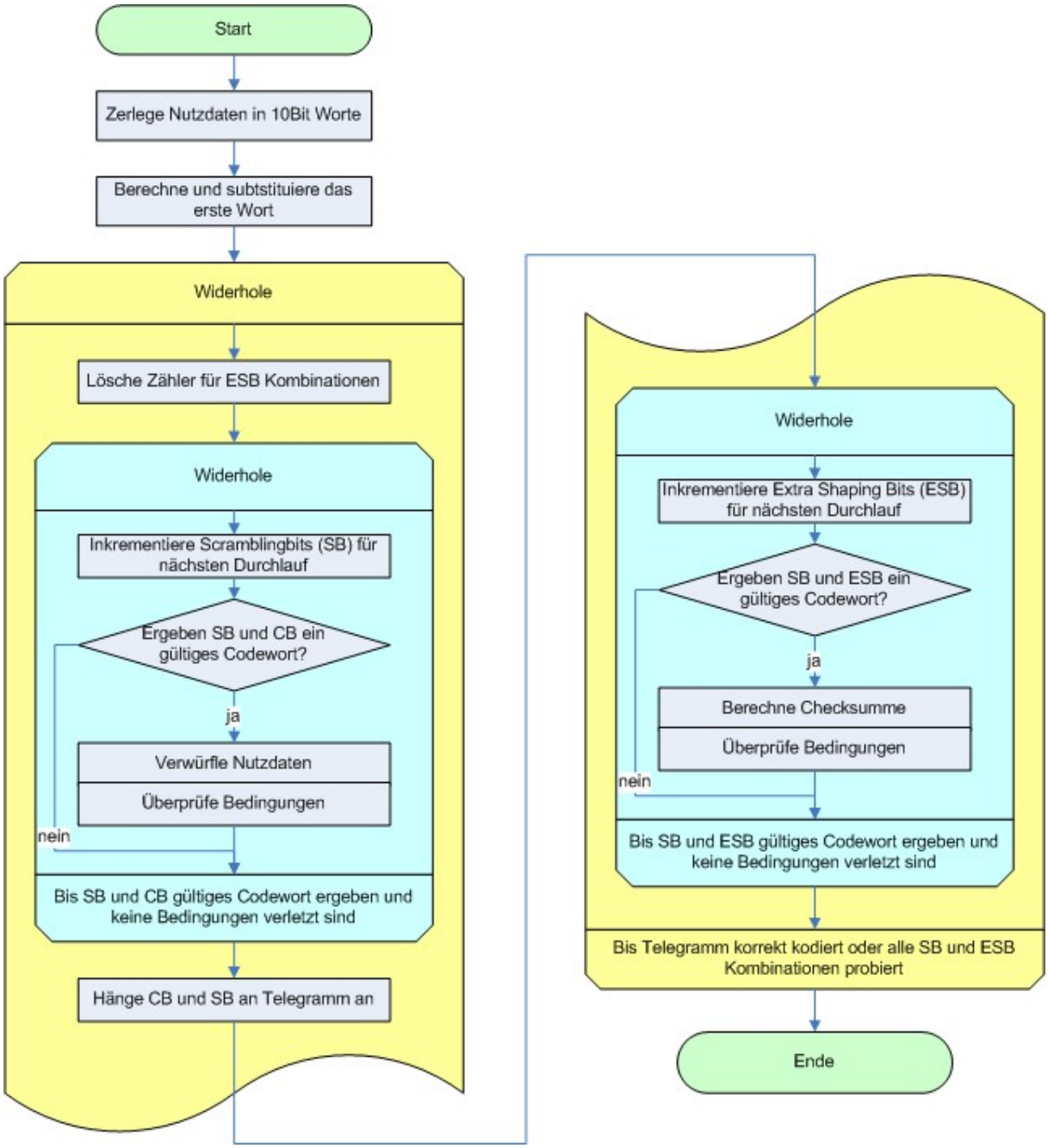

Abbildung 13: Flussdiagramm encodeTelegram (…)

4.2.2 Funktion srcambleAndSubst (…)

Diese Funktion verwürfelt die Nutzdaten im Telegramm und substituiert anschließend die verschlüsselten Worte durch die 11 Bit Worte aus der Substitutionstabelle. Für diesen Vorgang werden folgende Parameter an die Funktion übergeben:

Tabelle 6: Parameter der Funktion srcambleAndSubst (…)

Name	Typ	Beschreibung
words11	Uint16*	Zeiger auf den Array von 10 Bit Datenworten mit neuem höchsten Datenwort
words10	uint16*	Zeiger auf Array von 11 Bit Worten mit den fertig Verschlüsselten (srambled) Daten
count	uint16*	Anzahl der Worte im Telegramm
sb	uint8*	Pointer auf einen Array mit den Scramblingbits

Tabelle 7: Rückgabewert der Funktion srcambleAndSubst (…)

Name	Typ	Beschreibung
-	-	-

Der grundsätzliche Ablauf der Funktion lehnt sich an den fertigen Code zur Kodierung für das BTM/i des RailSiTe - Labors an und wird in Abbildung 14 näher erläutert.

Wie in Abbildung 14 zu sehen ist, wird zu Beginn der Funktion aus den Scramblingbits im Funktionsparameter **sb** der Initialwert des Schieberegisters **S** berechnet. Dazu dient folgende Formel:

$$S = (2801775573 \cdot < Scramblingbits >) \bmod 2^{32} \qquad \textbf{(3)}$$

Formel 3: Berechnung des Initialwertes des Schieberegisters S

Anschließend werden die Daten über eine Verknüpfung des Array mit den 10 Bit Benutzerdaten **word10[]** und dem Schieberegister in **S** zu einem neuen Array verknüpft. Dabei wird in den beiden Schleifen jedes Wort des Arrays **words10[]** einzeln verschlüsselt und in der Variable **si** zwischengespeichert.

Wurde ein komplettes 10 Bit Datenwort verschlüsselt, so wird es nicht in den Array zurückgespeichert, sondern sofort zur Bestimmung des Substitutionsworts benutzt. Dabei dient es als Index für den Array der Substitutionsworte. Das Ergebnis wird direkt an den Array für die fertig verschlüsselten 11 Bit Worte übertragen.

Am Ende dieser Funktion liegen alle Daten als 11 Bit Kodeworte im Array **words11[]** vor.

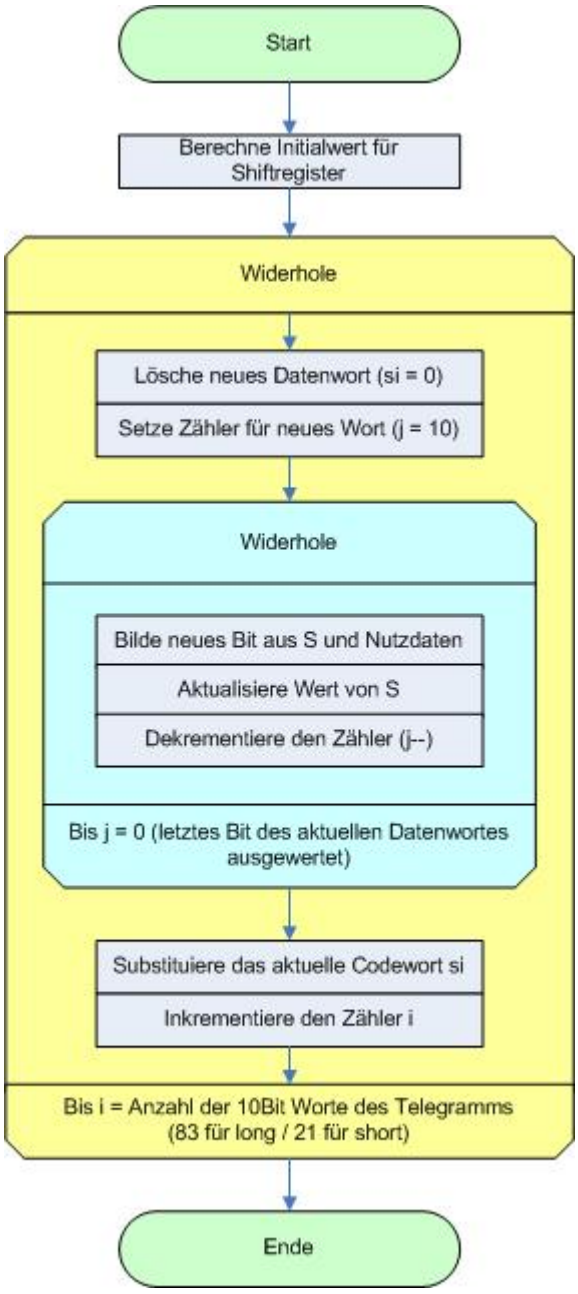

Abbildung 14: Flussdiagramm srcambleAndSubst (…)

4.2.3 Funktion polyRemainder (…)

Diese Funktion berechnet nach der Verschlüsselung der Daten den Rest der Polynomendivision, die für die Berechnung der Checksumme benötigt wird. Dazu müssen folgende Parameter an die Funktion übergeben werden:

Tabelle 8: Parameter der Funktion polyRemainder (…)

Name	Typ	Beschreibung
Remainder	Uint8	Zeiger auf das Bitfeld, das später den Rest enthalten soll (Ergebnis der Berechnung)
R	Uint16*	Zeiger auf die Variable, die nach der Berechnung die Länge des Polynoms der Berechnung enthalten wird
Poly1	Uint8*	Zeiger auf das Polynom 1 (Dividend)
Length1	Uint16	Länge des Polynoms 1
Poly2	Uint8*	Zeiger auf das Polynom 2 (Divisor)
Length2	Uint16	Länge des Polynoms 2

Tabelle 9: Rückgabewert der Funktion polyRemainder (…)

Name	Typ	Beschreibung
-	-	-

Der Algorhythmus zur Berechnung des Rests der Polynomendivision wurde komplett unverändert aus der Software für das BTM/i übernommen und wird hier nicht weiter erläutert.

4.2.4 Funktion checkConstraints (…)

Diese Funktion überprüft nach der abgeschlossenen Kodierung der Telegramme, ob alle Anforderungen, die ETCS an ein fertig kodiertes Balisentelegramm stellt, eingehalten werden. Sollte das nicht der Fall sein, so muss das Telegramm mit veränderten Voraussetzungen erneut kodiert werden.

Die Funktion hat folgende Parameter:

Tabelle 10: Parameter der Funktion checkConstraints (…)

Name	Typ	Beschreibung
Data	Uint8*	Zeiger auf den Array mit den fertig kodierten Daten als Bitfeld
Length	Uint16	Anzahl der Bits des Telegramms
Long_tlg	Bool	Ist wahr, wenn ein langes Telegramm kodiert werden soll, ansonsten False
Alphabet	Bool	Ist wahr, wenn die Alphabet Condition getestet werden soll, ansonsten False

Tabelle 11: Rückgabewert der Funktion checkConstraints (…)

Name	Typ	Beschreibung
-	-	-

Der Algorhythmus zur Berechnung des Restes der Polynomendivision wurde komplett unverändert aus der Software für das BTM/i übernommen und wird hier nicht weiter erläutert.

4.3 Performance Untersuchung

4.3.1 Ziel der Performance Untersuchung

Beim Test der fertig implementierten Kodierungsfunktionen auf dem DSP ist aufgefallen, dass die Dauer der Kodierung von Balisentelegrammen sehr stark von der Kombination der unkodierten Nutzdaten, der Sramblingbits (SB) und der Extra Shaping Bits (ESB) anhängig ist.

Für den Test wurden zwei unkodierte Telegramme und zwei SB / ESB Kombinationen als RS 232 Kommando erstellt, mit denen die Kodierungsfunktionen getestet wurden. Die Telegramme sind nachfolgend als hexadezimale Werte gegeben.

Tabelle 12: unkodiertes Telegramm 1 (Test Kodierung)

Byte Nr.	Daten									
0 – 9	80	14	00	05	6F	00	07	FF	FF	FF
10 – 19	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF
20 - 29	FF	FF	FF	FF	FF	FF	C0			

Tabelle 13: unkodiertes Telegramm 2 (Test Kodierung)

Byte Nr.	Daten									
0 – 9	DB	DD	B7	FF	BB	19	DC	EB	67	DA
10 – 19	79	51	BE	FF	DB	DD	B6	8F	FC	76
20 - 29	6D	76	FF	BF	BD	DB	00			

Tabelle 14: SB / ESB Kombination 1 (Test Kodierung)

Funktion	Wert binär	Wert hexadezimal
Scrambling Bits (SB)	0000 0000 0001 1100	0x001C
Extra Shaping Bits (ESB)	0000 0000 1101 1001	0x00D9

Tabelle 15: SB / ESB Kombination 2 (Test Kodierung)

Funktion	Wert binär	Wert hexadezimal
Scrambling Bits (SB)	0000 0001 0011 0010	0x0132
Extra Shaping Bits (ESB)	0000 0001 0010 0010	0x0122

Bei der Ausführung der zugehörigen RS 232 Kommandos wurde für die Dauer der Kodierung mit diesen Testtelegrammen folgende Zeiten gemessen:

Tabelle 16: Kodierungsdauern für Testtelegramme

Tlg. Nr.	SB / ESB Nr.	Kodierungsdauer
1	1	4,10 ms
1	2	28,00 ms
2	1	232,00 ms
2	2	2122,00 ms

Schon bei der Kodierung dieser wenigen Telegramme fällt die unterschiedliche Dauer der Kodierung auf. Am auffälligsten ist die Dauer der Kodierung des Telegramms 2 mit der SB / ESB Kombination 2. Da der DSP während der Dauer der Kodierung nicht für das RailSiTe - Labor erreichbar ist, ist es wichtig zu erfahren, ob diese Kombination zufällig eine besonders schlechte Kombination ist, oder dem Standard der Kodierungsdauer entspricht.

Um hierzu genauere Untersuchungen anstellen zu können, wird eine zusätzliche Funktion **getEncPerf (…)** implementiert. Diese Funktion ermöglicht es, Telegramme mit zufällig generierten Nutzdaten und zufällig ausgewählten Scrambling Bits und Extra Shaping Bits zu kodieren und dabei die Dauer der Kodierung zu messen. Die Funktion wird im nachfolgenden Kapitel genauer beschrieben.

4.3.2 Funktion getEncPerf (…)

Diese Funktion soll dem Test der Performance des Kodierungsalgorhythmus dienen. Sie wurde implementiert, um die Performance der Kodierung direkt in der DSP - Software testen zu können, ohne über externe Hilfsmittel Zeitmessungen machen zu müssen. Die Funktion **getEncPerf (…)** ermöglicht die Kodierung einer beliebigen Anzahl von Telegrammen mit beliebigem Format. Dabei werden die Nutzdaten über einen Zufallsgenerator erzeugt. Für die SB und ESB können entweder Initialwerte, oder zufällig generierte Kombinationen genutzt werden. Zu diesem Zweck verfügt die Funktion über folgende Parameter.

Tabelle 17: Parameter der Funktion getEncPerf (…)

Name	Typ	Beschreibung
cycles_u32	uint32	Anzahl der Telegramme, die für den Test kodiert werden sollen.
sb_init_u16	uint16	Startwerte für die Scrambling Bits bei der Kodierung der Telegramme. Der Wert 0xFFFF generiert für jedes Telegramm einen zufälligen Wert für die Scrambling Bits.
esb_init_u16	uint16	Startwerte für die Extra Shaping Bits bei der Kodierung der Telegramme. Der Wert 0xFFFF generiert für jedes Telegramm einen zufälligen Wert für die Extra Shaping Bits.
format_u8	uint8	Dieser Variable wird das Format der Telegramme übergeben, für das der Test durchgeführt werden soll. Dabei sind nur folgende Werte erlaubt: TLG_TYPE_SHORT: kurze Telegramme TLG_TYPE_LONG: lange Telegramme Alle anderen Werte führen zur Kodierung von langen Telegrammen.

Tabelle 18: Rückgabewert der Funktion getEncPerf (…)

Name	Typ	Beschreibung
-	-	-

Für jeden Testschritt, also jedes kodierte Telegramm wird die Nummer und die Dauer der Kodierung des Telegramms in Millisekunden als Klartextmeldung über den Diagnosekanal übertragen. Dadurch kann der Test über ein Terminalprogramm einfach mitgeschrieben werden, um ihn später statistisch auswerten zu können.

Zur Messung der Kodierungsdauer wird der Systemzähler in der Struktur **systime_s** benutzt. Dieser wird jede Millisekunde inkrementiert. Zur Ermittlung der Dauer der Kodierung wird der Zähler vor und nach der Kodierung ausgelesen. Die Differenz dieser beiden Werte bildet dann die Kodierungsdauer in Millisekunden. Besondere Obacht muss auf die Möglichkeit des Überlaufs des Zählers während der Kodierung gelegt werden. In diesem Fall muss die Differenz auf anderem Wege ermittelt werden.

Am Ende der Kodierung werden alle ermittelten Informationen über den Diagnosekanal an den Anwender gesendet. Darin enthalten ist die Anzahl der kodierten Telegramme, die Anzahl der korrekt und nicht korrekt kodierten Telegramme und Zeitinformationen zur gesamten, maximalen, minimalen und durchschnittlichen Kodierungsdauer pro Telegramm.

Die Funktion soll zusätzlich über den Diagnosekanal zugänglich sein, so dass der Anwender auf einfache Weise einen Performance Test des Kodierungsalgorhythmus starten kann.

Wichtig:

Für die Ermittlung der Zufallswerte in der Funktion wird die Standard C Funktion **rand (…)** benutzt. Diese Funktion ermittelt aber nur Pseudozufallszahlen. Diese treten nach einem Reset immer in der gleichen Abfolge auf. Aus diesem Grund sind alle Aussagen, die mit der Funktion **getEncPerf (…)** über die Performance des Kodierungsalgorhythmus gemacht werden, nur dann repräsentativ, wenn eine große Anzahl von Telegrammen pro Test kodiert wird. In diesem Fall ist davon auszugehen, dass alle Kombinationen, die zu Extrema führen können, im Test mit berücksichtigt werden.

Deshalb sind für die in den nachfolgenden Kapiteln gemachten Aussagen 10000 Telegramme kodiert worden.

4.3.3 Statistische Auswertung des Performance Tests

Für den Test der Performance des Encoders wurden mehrere Messreihen aufgenommen. Dabei wurden immer 1000 kurze Telegramme mit zufällig generiertem Inhalt kodiert. Als SB und ESB kamen dabei jeweils die SB / ESB Kombinationen 1 und 2, so wie auch zufällig generierte SB / ESB Kombinationen zum Einsatz. Die Ergebnisse der Messreihen können Abbildung 15, Abbildung 16 und Abbildung 17 entnommen werden.

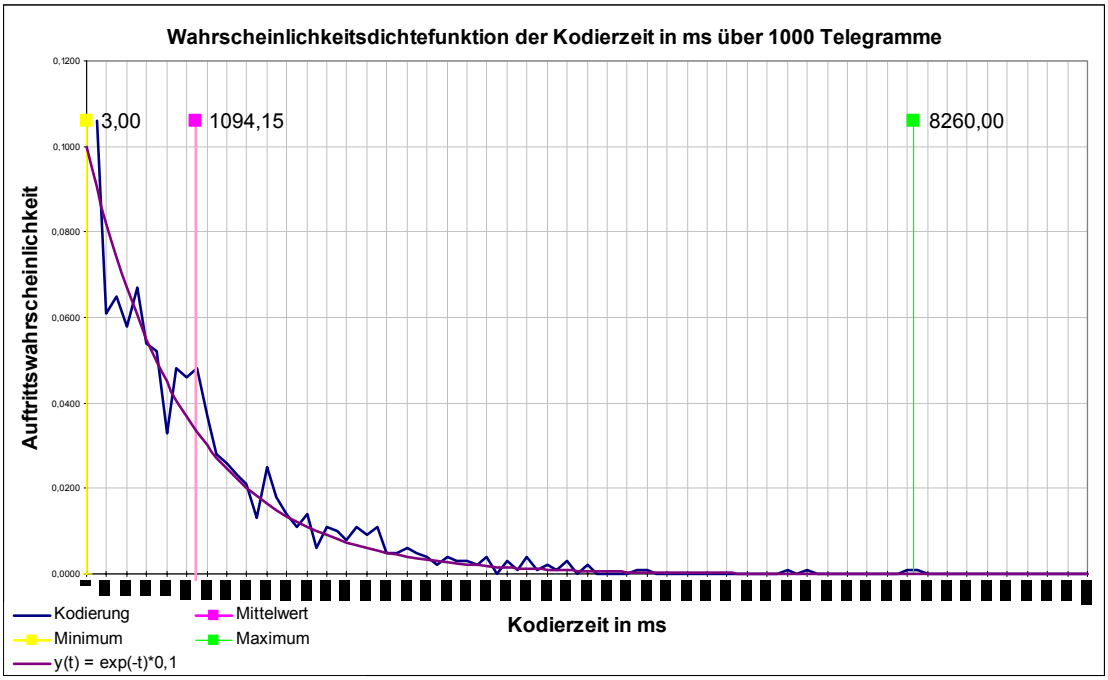

Abbildung 15: Statistik Kodierdauer mit zufälligen SB/ESB

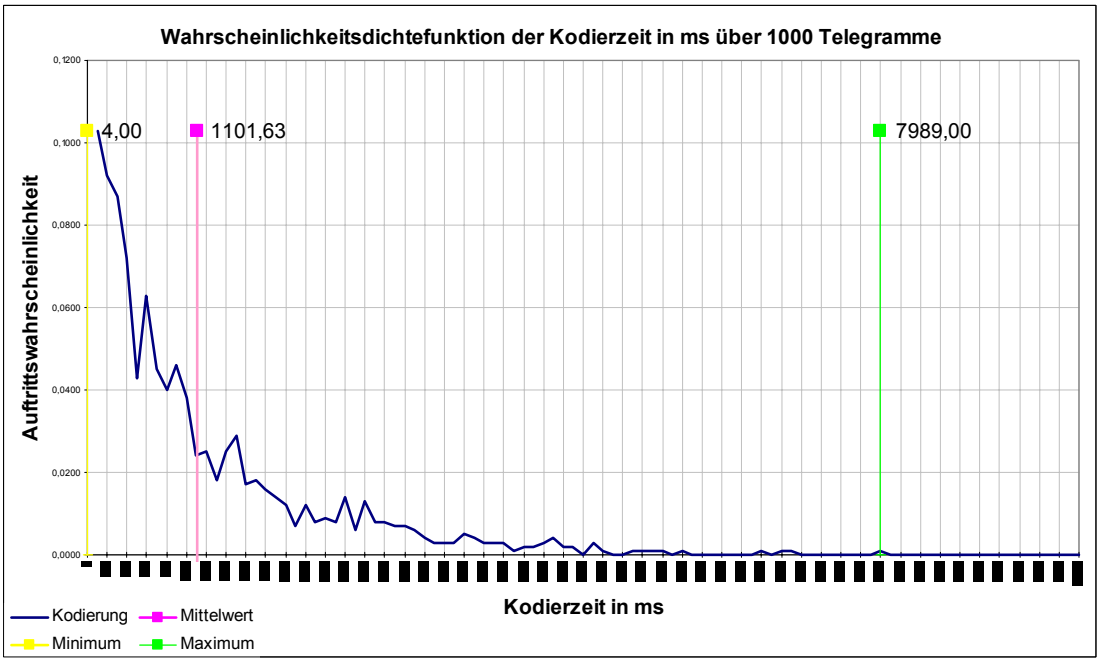

Abbildung 16: Statistik Kodierdauer mit SB/ESB Kombination-1

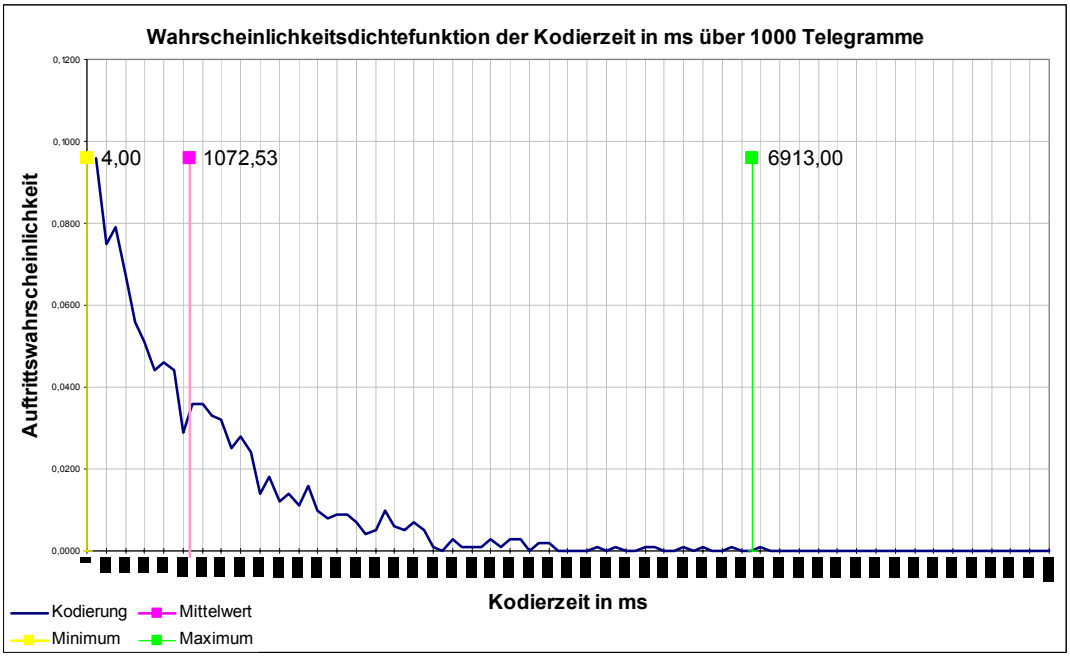

Abbildung 17: Statistik Kodierdauer mit SB/ESB Kombination-2

Wie in den Abbildungen zu sehen ist, ist der Verlauf in allen Diagrammen ähnlich. Die Dauer der Kodierung hängt also nicht nur von der Wahl der SB / ESB Kombinationen ab, wie man es aus den ersten Testreihen hätte vermuten können. Vielmehr ist die Dauer der Kodierung sehr stark abhängig von der Kombination von Nutzdaten und SB und ESB.

Aus diesem Grund wurde ein neuer Test durchgeführt, bei dem 10000 Telegramme kodiert wurden, um die Ergebnisse zu verifizieren. Außerdem sollten bei dieser großen Anzahl von Telegrammen und SB / ESB Kombinationen die Einflüsse des Pseudozufallsgenerators weitestgehend eliminiert werden. Die Ergebnisse dieses Tests können Abbildung 18 und Abbildung 19 entnommen werden.

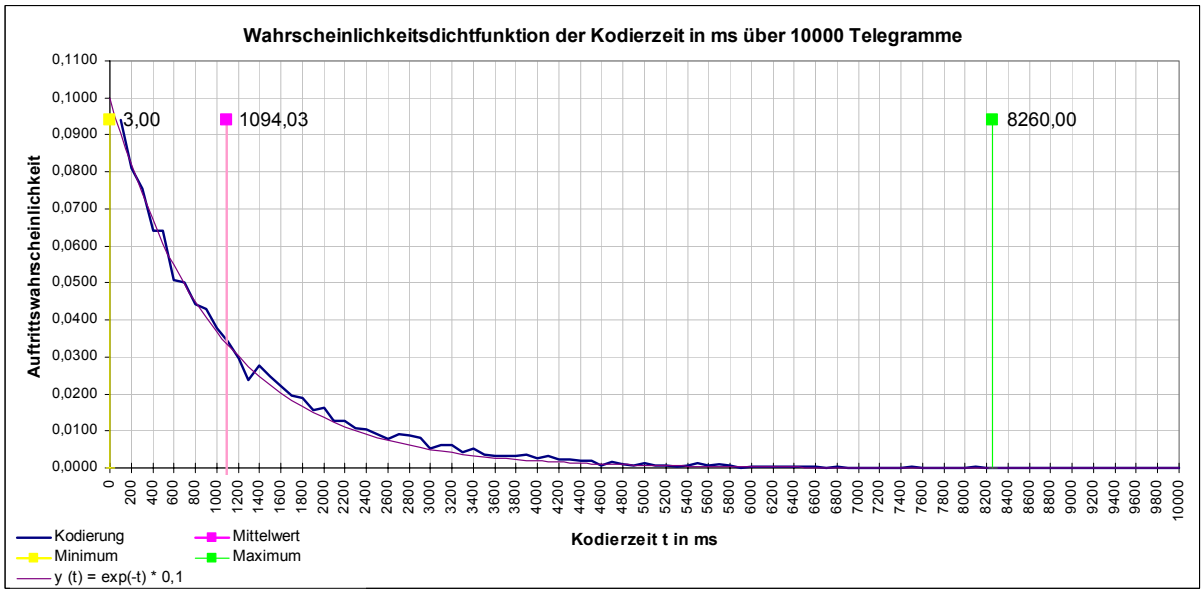

Abbildung 18: Wahrscheinlichkeitsdichtefunktion mit zuf. SB / ESB Komb. über 10000 Tlg.

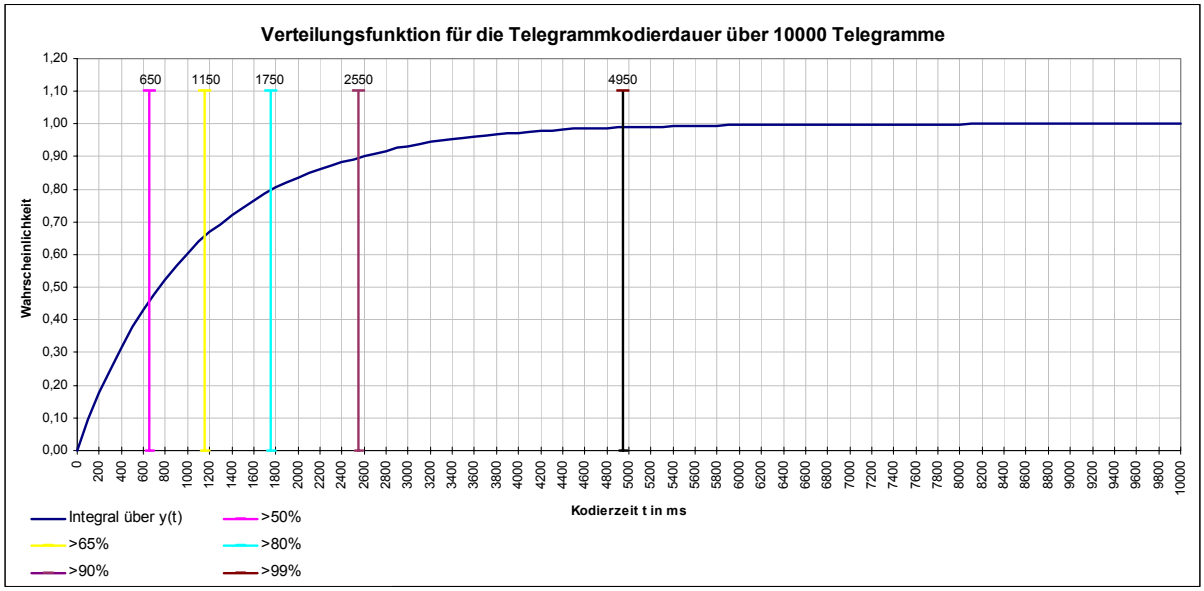

Abbildung 19: Verteilungsfunktion für 10000 Telegramme

Wie Abbildung 18 entnommen werden kann, beträgt die mittlere Kodierzeit ca. eine Sekunde. Über die Kodierungsdauer nimmt die Auftretenswahrscheinlichkeit expotentiell ab. Sie nähert sich dabei der Funktion $y(t) = 0{,}1 \cdot e^{-t/\tau}$ mit $\tau = 1/s$ an.

Aus diesen Daten lassen sich Forderungen an das RailSiTe - Labor bezüglich der Zeit stellen, die dem DSP für die Kodierung der Telegramme gegeben werden muss. Dazu wurde in Abbildung 19 numerisch das Integral über die Auftretenswahrscheinlichkeit berechnet. Wie zu erkennen ist, muss die Zeitdauer abhängig von der Wahrscheinlichkeit gewählt werden, mit der die Telegramme innerhalb dieses Zeitfensters kodiert sein müssen. Diese Zeitdauer wird mit zunehmender Wahrscheinlichkeit sehr groß. Exemplarisch wurden folgende Werte ermittelt.

Tabelle 19: Mindestwartezeiten für RaiSiTe Labor pro zu kodierendem Telegramm

p für $T_{Kodier} < T_{Grenze}$	T_{Grenze}
> 50%	> 650 ms
> 65%	> 1150 ms
> 80%	> 1750 ms
> 90%	> 2550 ms
> 99%	> 4950 ms

Zum Beispiel ist die Wahrscheinlichkeit, dass alle zu kodierenden Telegramme unter einem Grenzwert T_{Grenze} liegen bei einer oberen Kodierdauer von T_{Grenze} = 1750 ms gleich 80 %. Daraus kann eine Mindestwartezeit für das RailSiTe abgeleitet werden, die es dem DSP pro zu kodierendem Telegramm lassen muss.

Da diese Zeit unter Umständen zu lang ist, um das RailSiTe - Labor warten zu lassen, wurde eine Strategie entwickelt, mit der die Kodierung der Telegramme in einen weiteren Task ausgelagert werden kann, der parallel zur restlichen DSP - Software läuft. Die Abarbeitung von Sequenzen und das Speichern von Telegrammen im DSP ist dann weiterhin auch während der Kodierung der Telegramme möglich.

Als zweite Möglichkeit kann der Optimizer (Werkzeug zur Optimierung des Quellkodes auf dem DSP) des Code Composer Studio aktiviert werden. Dabei müssen die Auswirkungen der Optimierungen auf das Programm überprüft werden.

Die Messergebnisse nach einer Optimierung des Quellcodes und das genaue Vorgehen bei einer Umstrukturierung der Firmware werden in den nächsten Kapiteln genauer beschrieben.

4.3.4 Performancegewinn durch Optimierung

Das Code Composer Studio bietet in unserer Version die Möglichkeit, den Quellcode nach der Kompilierung durch einen Optimizer auf die Zielarchitektur des TSM320C6713 zu optimieren. Als echter DSP ist dieser in der Lage mehrere Befehle parallel auszuführen. Außerdem hat er die Möglichkeit, den Befehlsdurchsatz durch Einsatz einer Software Pipeline stark zu erhöhen. Zu diesem Zweck bietet das Code Composer Studio (CCS) fünf verschiedene Einstellmöglichkeiten für den Optimizer. Diese sind:

- ⇨ **„none“** – keine Optimierung des Quellcodes:

  In dieser Einstellung wird der Quellcode lediglich in Assembler übersetzt. Der Compiler führt keine Optimierungen des Quellcodes durch.

- ⇨ **„register“** – Vermeidung unnötiger Speicherzugriffe:

  In dieser Einstellung wird versucht, möglichst alle Speicherzugriffe und Zugriffe auf Variablen über die internen Register zu machen, da diese sehr viel schneller angesprochen werden können, als Adressen im Speicher.

- ⇨ **„local“** – Optimierung des Zugriffes größerer Kodeblöcke:

  Es wird versucht, größere Kodeblöcke zu optimieren.

- ⇨ **„function“** – Optimierung einer ganzen Funktion:

  In dieser Einstellung versucht der Optimizer den Quellcode einer ganzen Funktion zu optimieren. Ab dieser Einstellung ist die Software Pipeline des DSP aktiv. Der Optimizer versucht daher Schleifen in parallele Strukturen umzuwandeln, die dann sehr viel weniger Durchläufe erfordern.

- ⇨ **„file“** – Optimierung einer ganzen Sourcedatei:

  In dieser Einstellung versucht der Optimizer den Quellcode einer ganzen Datei zu optimieren. Sie bringt das höchste Maß an Geschwindigkeitsgewinn, bietet aber die Gefahr, dass der optimierte Quellcode nicht mehr dem eigentlichen Quellcode entspricht.

In verschiedenen Tests mit den Optimierungseinstellungen ist aufgefallen, dass die Firmware mit den beiden höchsten Optimierungseinstellungen, nicht mehr problemlos funktioniert. Andererseits bieten diese Einstellungen den größtmöglichen Geschwindigkeitszuwachs, gerade für die performancekritische Kodierung der Balisentelegramme.

Um die Vorteile des Optimizers zu nutzen, aber trotzdem alle Komponenten, wie z.B. den UART zu lauffähigem Quellcode zu kompilieren, wird der Optimierungslevel für alle Quelldateien auf „local“ gestellt. In dieser Einstellung ist das Softwarepipelining, das besonders bei Endlosschleifen, wie in der UART ISR vorhanden, zu Problemen geführt hat, deaktiviert.

Für die Funktionen zur Kodierung der Balisentelegramme in der Datei **encode_functions.c** wird die Einstellung des Optimizers allerdings auf „file“ gesetzt. Dadurch entsteht ein lauffähiges Programm mit stark optimiertem Kodierungsalgorhythmus. Die Performanceergebnisse nach der Aktivierung des Optimizers sind in den nachfolgenden Diagrammen dargestellt.

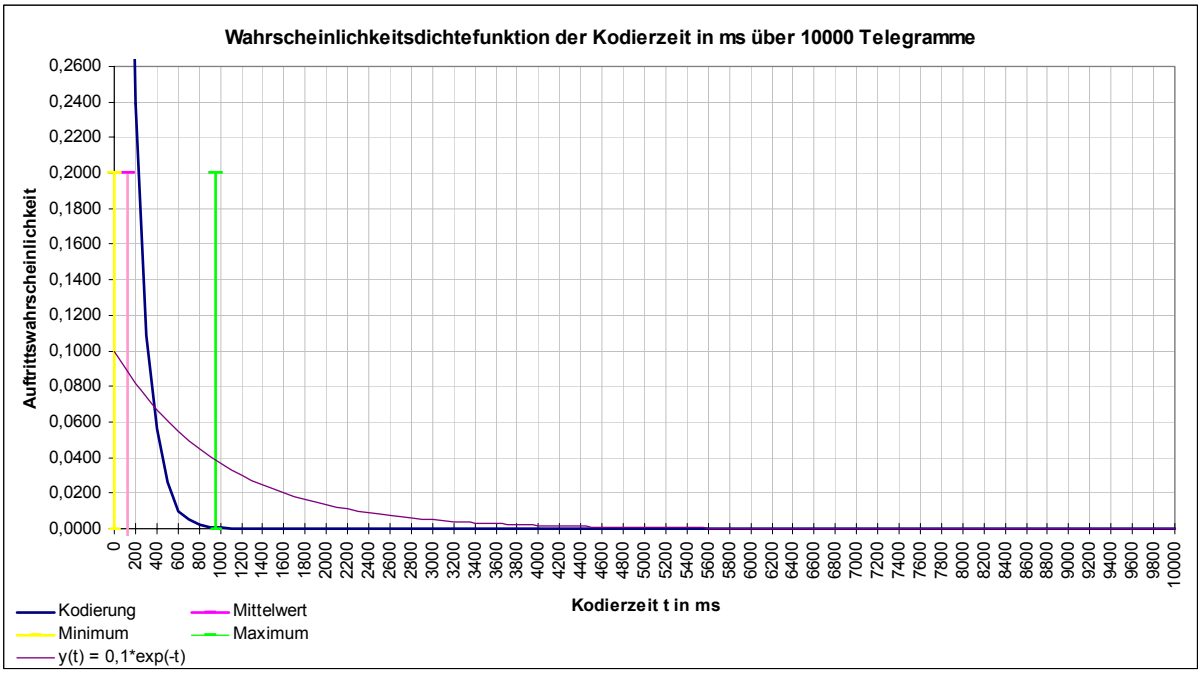

Abbildung 20: Kodierdauer mit zufälliger SB / ESB über 10000 Telegramme (short)

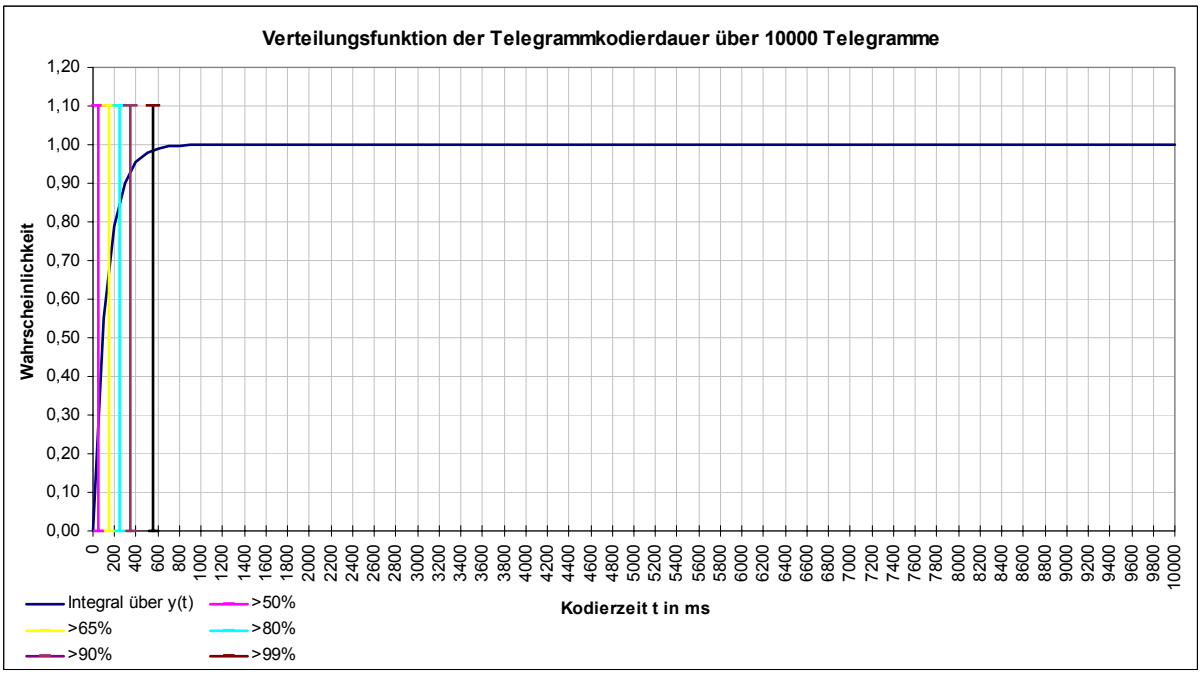

Abbildung 21: Integral über Auftretenswahrscheinlichkeit für 10000 Telegramme (short)

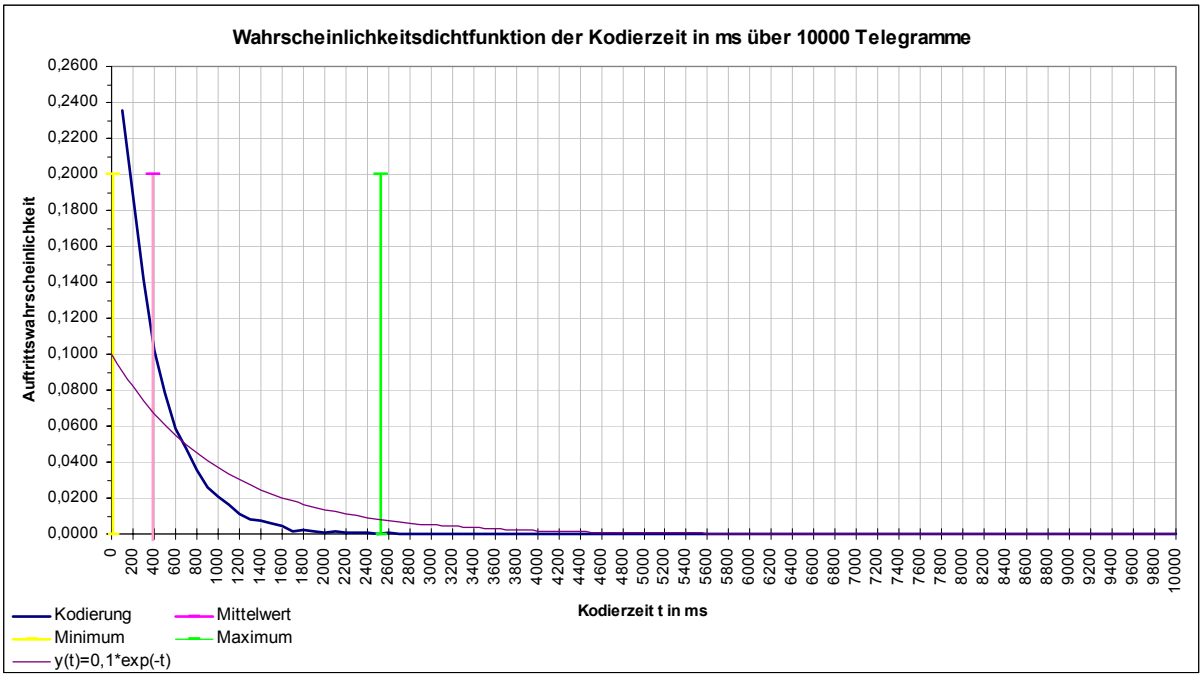

Abbildung 22: Kodierdauer mit zufälliger SB / ESB über 10000 Telegramme (long)

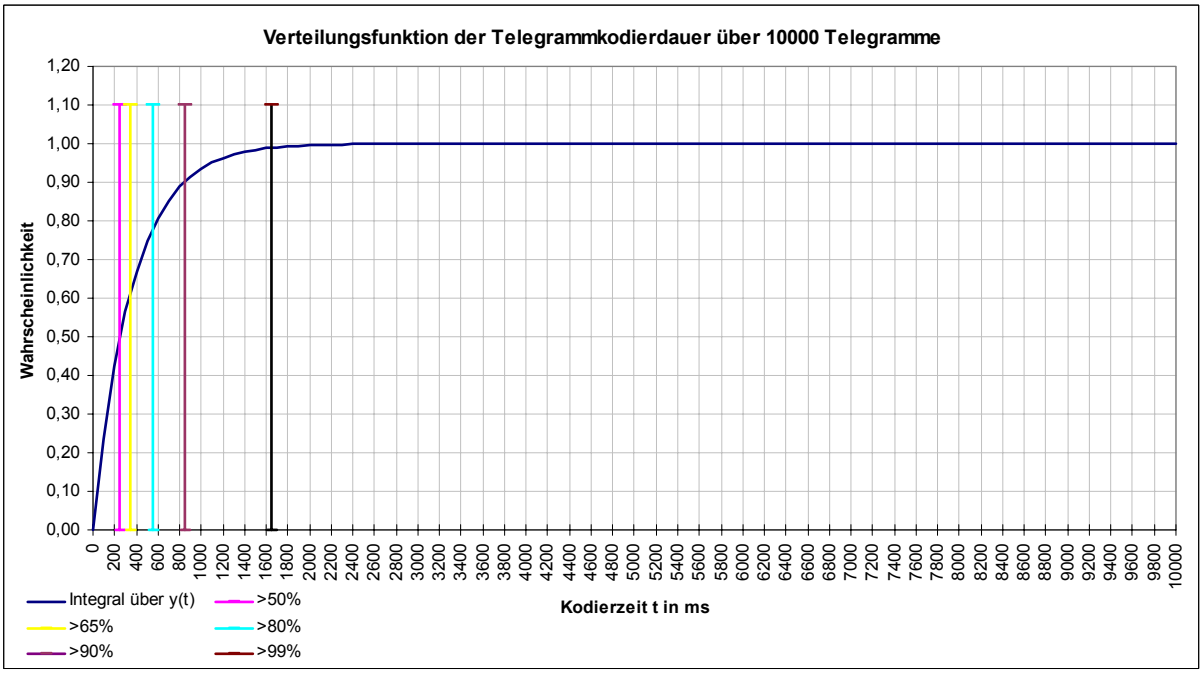

Abbildung 23: Integral über Auftretenswahrscheinlichkeit für 10000 Telegramme (long)

Wie in den Abbildung 20 bis Abbildung 23 zu sehen ist, ist die Performance des Kodierungsalgorhythmus durch die Optimierung stark gestiegen. Zum Vergleich ist die Funktion $y(t) = 0{,}1 \cdot e^{-t/\tau}$ auch hier mit dargestellt, an die sich die Kodierung der kurzen Telegramme vor der Aktivierung des Optimizers angenähert hatte.

In Abbildung 20 ist zu erkennen, dass der Verlauf der Auftretenshäufigkeit jetzt über die größten Bereiche unterhalb von y(t) liegt. Die folgende Tabelle soll den Gewinn an Leistung veranschaulichen.

Tabelle 20: Leistungsgewinn der Kodierung nach Optimierung (kurze Tlg.)

p für $T_{Kodier} < T_{Grenze}$	T_{Grenze} (ohne Opt.)	T_{Grenze} (mit Opt.)	Gewinn in %
> 50%	> 650 ms	> 50 ms	92,3 %
> 65%	> 1150 ms	> 150 ms	86,9 %
> 80%	> 1750 ms	> 250 ms	85,7 %
> 90%	> 2550 ms	> 350 ms	86,3 %
> 99%	> 4950 ms	> 550 ms	88,9 %

Die jetzige Performance der Kodierung sollte ausreichen. Für kurze Telegramme kann jetzt mit einer Wartezeit von 550 ms zu 99% sichergestellt werden, dass das Telegramm innerhalb dieser Wartezeit kodiert werden kann. Für lange Telegramme beträgt diese Wartezeit nur noch 1650 ms. Von weiteren manuellen Optimierungen des Quellcodes, wie z.B. ersetzen der C Funktionen durch eigene Assembler Routinen, wird deshalb abgesehen.

Sollte trotzdem die Notwendigkeit zur Erweiterung der Firmware auf mehrere parallele Tasks bestehen, ist im nächsten Kapitel ein Lösungsansatz hierzu gegeben.

4.3.5 Alternative Firmwarestruktur der Kodierung in separatem Task

Die hier angestrebte Struktur der DSP - Software strebt die Ausführbarkeit der Kodierung in einem separaten Task (engl. Aufgabe) an, der quasi parallel zur übrigen Firmware im DSP abläuft. Der große Vorteil dieses Vorgehens wäre, das der DSP selbst während einer Kodierung für das RailSiTe - Labor und auch Anwender über den Diagnosekanal voll erreichbar ist.

Nachteilig ist der große Programmierungs- und Schnittstellenaufwand, da zwei parallele Tasks über definierte Schnittstellen miteinander kommunizieren müssen. Sollte eine solche Struktur erforderlich sein, so muss der gesamte Kodierungsalgorhythmus auf dem DSP in eine separate Funktion, verlagert werden, in der die gesamte Kodierung abläuft. An diese Funktion werden spezielle Anforderungen gestellt, um sie in einem zyklischen Task abarbeiten zu können.

⇨ **Ausführbarkeit in Teilschritten definierter Länge (Dauer):**

Damit die Kodierung in einem parallelen Task zur restlichen DSP - Software laufen kann, müssen die Funktionen zur Kodierung so umgestaltet werden, dass die gesamte Kodierung eines Telegramms in einzelne Teilstücke mit definierter maximaler Dauer zerlegt werden können.

Diese maximale Zeit pro Aufruf der Kodierungsfunktion darf nicht überschritten werden, da sie sonst vor dem nächsten Aufruf noch nicht beendet ist (zeitlicher Überlauf). In diesem Fall bliebe auch keine Rechenzeit zur Ausführung der restlichen Firmware übrig.

⇨ **Einhaltung festgelegter Schnittstellen:**

Um die Kommunikation der Kodierung mit der restlichen DSP - Software zu ermöglichen, müssen einheitliche Schnittstellen definiert werden. Besonders ist darauf zu achten, dass keine Schnittstellen entstehen, die sowohl von der Kodierung, als auch der restlichen Software gleichzeitig bearbeitet werden, da auf diese Weise undefinierte Zustände entstehen können.

Für die parallele Ausführung der Kodierung wird die Funktion **encodeTelegram (…)** nach den oben genannten Gesichtspunkten umgestaltet. Anschließend wird sie zyklisch aus der Funktion **timer0_ISR (…)** aufgerufen. In dieser alle 1 ms aufgerufenen Funktion kann der Aufrufzyklus der Kodierung durch einen Zähler festgelegt werden. Die Funktion **encodeTelegram (…)** wird dann nur alle X Aufrufe der Funktion **timer0_ISR (…)** aufgerufen.

Um den zyklischen Aufruf der Funktion **encodeTelegram (…)** zu ermöglichen, wird ihre Struktur in eine Statemachine umgewandelt. Dabei sollte jeweils die Verwürfelung (scrambling) und die Berechnung der Checksumme (remainder) einem eigenen State entsprechen, der einmal, oder mehrfach pro Aufruf **encodeTelegram (…)** durchlaufen wird. Alle Variablen müssen für diesen Fall global definiert werden, damit ihre Werte nach dem Ende einer Funktionsperiode erhalten bleiben.

Für die Kommunikation der Funktion **encodeTelegram (…)** mit der restlichen Software (vor allem RailSiTe - Protokoll) werden einheitliche Schnittstellen geschaffen. Hierzu sind folgende Änderungen der vorhandenen DSP - Software notwendig:

⇨ **Zusätzliche Boolean Variable in allen Telegrammspeicherplätzen (tlgHeap)**

In allen Telegrammspeicherplätzen muss eine zusätzliche Variable angelegt werden, die angibt, ob das Telegramm gerade kodiert wird. Diese Variable wird nur von der Funktion **encodeFunctions (…)** gesetzt, oder gelöscht. Wenn sie *True* ist, darf das Telegramm im zugehörigen Speicherplatz weder überschrieben, noch in einer Sequenz gesendet werden

⇨ **Neue Struktur zur Kommunikation RailSiTe Protokoll, Kodierung**

Es wird eine neue Struktur angelegt, über die die Funktion **handle_RS2DSP _EncodeTlg (…)** mit der Funktion **encodeTelegram (…)** kommuniziert. Sie hat folgende Form:

```
Typedef struct
{
    Uint16 enc_tlg_nr_u16;    // Nummer des Telegrammspeicherplatz für Kodierung
    Uint8 enc_func_state_u8; // State, der Statemachine für encodeTelegram(…)
    Bool enc_running_u8;      // True, wenn gerade kodiert wird
    Uint16 enc_result_u16;    // Ergebnis der Kodierung
} t_encoder_status
```

Dabei wird die Variable **enc_tlg_nr_u16** nur von der Funktion **handle_RS2DSP_EncodeTlg (…)** beschrieben, während alle anderen Variablen der Struktur nur von der Funktion **encodeTelegram (…)** beschrieben werden.

⇨ **Neuer Parameter für handle_RS2DSP_EncodeTlg (…)**

Die Funktion **handle_RS2DSP_EncodeTlg (…)** muss dahingehend geändert werden, dass sie einen neuen Parameter erhält, über den angegeben werden kann, was das Kommando **RS2DSP_EncodeTlg (…)** bewirken soll. Folgende Optionen stehen zur Verfügung:

1. Kodierung für Telegramm starten
2. aktuelle Kodierung abbrechen
3. Status der aktuellen Kodierung lesen

Diese Unterscheidung ist notwendig, da die Funktion **handle_RS2DSP_EncodeTlg (…)** ansonsten erst verlassen wird, wenn die Kodierung abgeschlossen ist und auch erst dann ein ACK an das RailSiTe sendet. Zu diesem Zweck müssen auch neue ACK Meldungen definiert werden, die diese Stati an das RailSiTe zurückmelden können. Eine Abbruchbedingung ist notwendig, da die Kodierung in einem eigenen Task unter Umständen sehr lange dauern kann.

Wichtig:

Grundsätzlich werden bei diesem Vorgehen alle Kodierungen ausgebremst. Auch Kodierungen von Telegrammen, die ansonsten sehr schnell abgearbeitet sind brauchen mindestens drei bis vier Aufrufe der Funktion **encodeTelegram (…)**. Teilweise können die Kodierungen extrem lange dauern.

Bei der angedachten Zerlegung der Funktion **encodeTelegram (…)** nach den vorhandenen Schleifen kann bei einer Aufrufperiode der Funktion **encodeTelegram (…)** von 50ms im schlechtesten Fall folgende Dauer für die Kodierung entstehen:

$$2^{12}\langle sb\_Kombinationen\rangle \cdot 2^{10}\langle esb\_Kombinationen\rangle = 2^{24}\langle Loops\rangle \qquad \textbf{(4)}$$

$$2^{24}\langle Loops\rangle \cdot 50\frac{ms}{Loop} = 9{,}71\langle Tage\rangle$$

Formel 4: Dauer der Kodierung (worst case)

Um die Kodierung so schnell wie möglich zu machen, sollte die Aufrufperiode von **encodeTelegram (…)** so klein wie möglich gewählt werden. Die Periodendauer ist allerdings nach unten begrenzt, da bei zu kleinen Werten keine Zeit mehr für die Abarbeitung der restlichen DSP - Software bleibt.

Die Periode muss zumindest so groß gestaltet werden, dass innerhalb der Restrechenzeit in einer Periode kein Überlauf des Empfangspuffers im UART auftreten darf. Ansonsten gehen dem DSP - Daten verloren und RailSiTe - Labor und DSP laufen nicht mehr synchron.

5 Fehlersimulation in Telegrammen

5.1 Einführung

Bei jeder Übertragung von Daten über einen beliebigen Kanal können Fehler auftreten, die die Daten für den Empfänger möglicherweise unbrauchbar machen. Diese Fehler können unterschiedlichste Ursachen und Formen haben, führen aber immer dazu, dass die Daten beim Empfänger verfälscht vorliegen.

Über eine gute Kanalkodierung kann der Einfluss von Fehlern auf einer Übertragungsstrecke vermindert und in manchen Fällen sogar beseitigt werden. Dadurch werden die Daten mit größerer Übertragungssicherheit an den Empfänger übermittelt. Wichtiger ist jedoch, dass der Empfänger durch die Kanalkodierung überhaupt erst feststellen kann, ob Daten fehlerhaft sind, oder korrekt übertragen wurden.

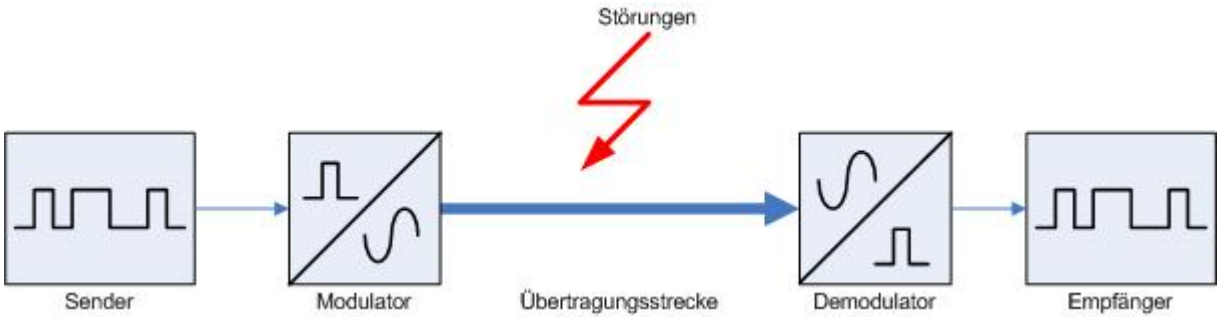

Abbildung 24: Prinzipdarstellung Übertragungsstrecke

Gerade bei sicherheitsrelevanten Daten, wie sie in den Balisentelegrammen der ETCS Zugleittechnik vorkommen, muss auf den möglichen Einfluss von Fehlern in der Übertragungskette der Daten besondere Rücksicht genommen werden.

Um den Fehlereinfluss zu minimieren, wird hier eine besondere Methode der Kanalkodierung verwendet, deren Aufbau und Umsetzung in Kapitel 4 bereits erläutert wurde.

Um die tatsächliche Sicherheit gegen Übertragungsfehler der Balisentelegramme testen zu können, wird auf dem DSP - Board ein zusätzliches RS 232 Kommando eingerichtet, mit dem gezielt Fehler in die Telegramme und deren Übertragung eingebracht werden können.

In den nachfolgenden Kapiteln sollen die einzelnen Fehlermodelle, die auf dem DSP simuliert werden können, genauer erläutert werden.

5.2 Mögliche Fehlermodelle zur Simulation auf dem DSP

Auf den meisten Übertragungsstrecken werden Daten (auch digitale Daten) über einen analogen Kanal übertragen. Bei dieser Übertragung können Effekte wie Übersprechen (Übersprechen eines zweiten analogen Kanals in die Übertragung) und Interferenzeffekte, z.B. durch Reflektionen entstehen.

Diese analogen Fehlereinflüsse können allerdings mit dem DSP - Board nicht nachgebildet werden, da zur Datenübertragung nur eine Antenne zur Verfügung steht. Außerdem sind solche Effekte durch die vorgeschriebene Art der Installation der Eurobalisen im Gleisbett weitestgehend unmöglich, da die Feldausbreitung der Eurobalisen so spezifiziert ist, dass es selbst bei der Anbringung zweier Balisen in benachbarten Gleisen und gleichzeitiger Anregung nicht zu Übersprechen kommen kann.

Deshalb beschränkt sich die Fehlersimulation in der DSP - Software auf Fehler, die allgemein bei der Übertragung von digitalen Daten auftreten können. Diese Fehler und deren Einfluss auf die Datenübertragung werden nachfolgend genauer beschrieben.

Dabei wird in vielen Fällen von einem Telegrammfenster T_{Window} gesprochen, in dem sich eine bestimmte Anzahl von Fehlern befinden darf, um noch sicher vom Empfänger erkannt werden zu können. Dieses Telegrammfenster ist wie folgt definiert:

$$T_{window} = (n + r) \cdot bit \qquad \textbf{(5)}$$

mit

n = Anzahl der Bits im Telegramm (short: n = 341, long: n = 1023) und

r = Telegrammüberlapp für Formaterkennung im Empfänger (short: r = 121, long: r = 77)

Formel 5: Definition des Telegrammfensters T_{Window}

5.2.1 Einzelbitfehler

Bei Einzelbitfehlern werden einzelne Bits der Datenübertragung verfälscht, also in ihrem Wert invertiert. Einzelbitfehler treten meist stochastisch Verteilt innerhalb eines Datenstroms auf. Ihre Vorkommenshäufigkeit hängt dabei stark von der Auslastung des Kanals und den Störeinflüssen ab.

Durch Zugabe von Redundanz in einen Kode sind die meisten Übertragungskanäle in der Lage, Einzelbitfehler zuverlässig zu erkennen. Bei der Verwendung von FEC (Forward Error Correction) Kodes können diese Einzelbitfehler oft sogar beim Empfänger korrigiert werden.

Die Anzahl der in einem einzelnen Kodewort erkennbaren Einzelbitfehler hängt direkt von der Hammingdistanz der Kodeworte ab. Sie berechnet sich wie folgt:

Seien zwei Kodeworte der Länge N über dem Alphabet A als Vektoren gegeben:

$$\vec{a} = (a_1, a_2, ..., a_N)^T \text{ und } \vec{b} = (b_1, b_2, ..., b_N)^T$$

Dann wird die folgende natürliche Zahl als Hamming Distanz d der beiden Kodeworte bezeichnet:

$$d(\vec{a}, \vec{b}) = \left| \left\{ i \middle| a_i \neq b_i \right\} \right| \qquad \textbf{(6)}$$

Die Hamming Distanz d ist also die Anzahl der Komponenten von $\vec{a}$ und $\vec{b}$, die nicht identisch sind.

Formel 6: Definition der Hamming Distanz d

Besondere Bedeutung kommt hier der minimalen Hamming Distanz d_{min} eines Kodes zu. Sie beschreibt die minimale Hamming Distanz aller Kodeworte des Kodes. Aus ihr lassen sich Aussagen für die Fähigkeit der Fehlererkennung und Fehlerkorrektur des Kodes treffen. Es gilt:

Für einen beliebigen Kode mit der minimalen Hamming Distanz d_{min} können immer

$$N_{FE} = d_{\min} - 1 \qquad \textbf{(7)}$$

Einzelbitfehler sicher erkannt und

$$N_{FK} = \frac{d_{\min}}{2} - 1 \text{ für } d_{min} \text{ gerade}$$

$$N_{FK} = \frac{(d_{\min} - 1)}{2} \text{ für } d_{min} \text{ ungerade}$$

Einzelbitfehler korrigiert werden.

Formel 7: Definition Anzahl der erkennbaren und korrigierbaren Einzelbitfehler

Für den Kode der Balisentelegramme nach dem ETCS Standard ist die minimale Hammingdistanz für lange Telegramme $d_{min} = 15$ und für kurze Telegramme $d_{min} = 17$ innerhalb eines Telegrammfensters T_{Window}. Es können also bis zu 16 Einzelbitfehler sicher erkannt werden.

Daher sollte auf dem DSP die Möglichkeit bestehen, bis zu 20 Einzelbitfehler in das Telegramm einbringen zu können. Dabei ist darauf zu achten, dass sowohl gerade, als auch ungerade Anzahlen von Einzelbitfehlern innerhalb eines Telegramms simuliert werden können, da manche Kodes gerade und ungerade Anzahlen von Einzelbitfehlern unterschiedlich gut erkennen können. Folgende Schrittweite wird dafür vorgeschlagen:

Tabelle 21: Einstellmöglichkeiten zur Simulation von Einzelbitfehlern

Nr.	Beschreibung
1	Einfügen von einem Bitfehler in einem Telegramm
2	Einfügen von zwei Bitfehler in einem Telegramm
3	Einfügen von drei Bitfehler in einem Telegramm
4	Einfügen von vier Bitfehler in einem Telegramm
5	Einfügen von fünf Bitfehler in einem Telegramm
7	Einfügen von sieben Bitfehler in einem Telegramm
10	Einfügen von zehn Bitfehler in einem Telegramm
14	Einfügen von vierzehn Bitfehler in einem Telegramm
15	Einfügen von fünfzehn Bitfehler in einem Telegramm
16	Einfügen von sechzehn Bitfehler in einem Telegramm
17	Einfügen von siebzehn Bitfehler in einem Telegramm
20	Einfügen von zwanzig Bitfehler in einem Telegramm

5.2.2 Bitfehlerbursts

Bei Bitfehlerbursts (engl. Burst = Anhäufung) handelt es sich um Anhäufungen von Bitfehlern, die nicht vereinzelt, sondern in Gruppen auftreten. Dabei wird keinesfalls jedes einzelne Bit innerhalb eines Bursts invertiert. Fehlerbursts bestehen vielmehr aus einer Gruppe von Bits gleichen Wertes. Es wird also eine ganze Gruppe von Bits entweder gelöscht, oder gesetzt. Ein Beispiel ist in Abbildung 25 zu sehen.

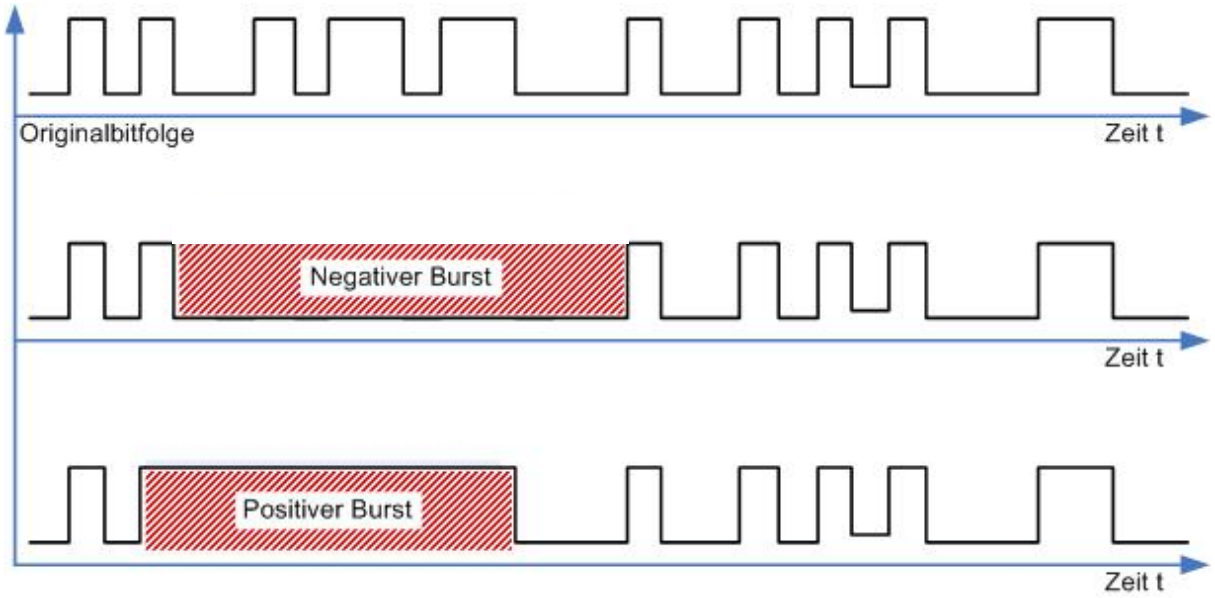

Abbildung 25: Beispiel für Bitfehlerbursts

Bitfehlerbursts treten immer dann auf, wenn ein Datenkanal kurzzeitig einer starken Störung unterliegt. Ein Beispiel hierfür sind große EMV - Spikes in elektrischen Maschinen (z.B. elektrische Motoren bei Bremsvorgängen), oder Pollingvorgänge bei Mobiltelefonen.

In den meisten Kodes, die für die Übertragung von digitalen Daten verwendet werden, werden zur Vermeidung solcher Fehlerbursts Verwürfler (engl. Interleaving, Scrambler) eingesetzt. Sie sorgen dafür, das Fehlerbursts nach der Verwürflung in Einzelbitfehler umgewandelt werden die wesentlich besser zu handhaben sind. Im Normalfall werden die Bits der Datenworte hierzu einfach durcheinander gewürfelt (engl. Scrambled) und so zu neuen Kodeworten zusammengesetzt. Wenn dann auf der Übertragungsstrecke Bitfehlerbursts auftauchen, werden diese beim Empfänger durch das Zurücktauschen der Bits (engl. descrambling) zu Einzelbitfehlern umgewandelt. Dieses Vorgehen in stark vereinfachter Form zeigt Abbildung 26.

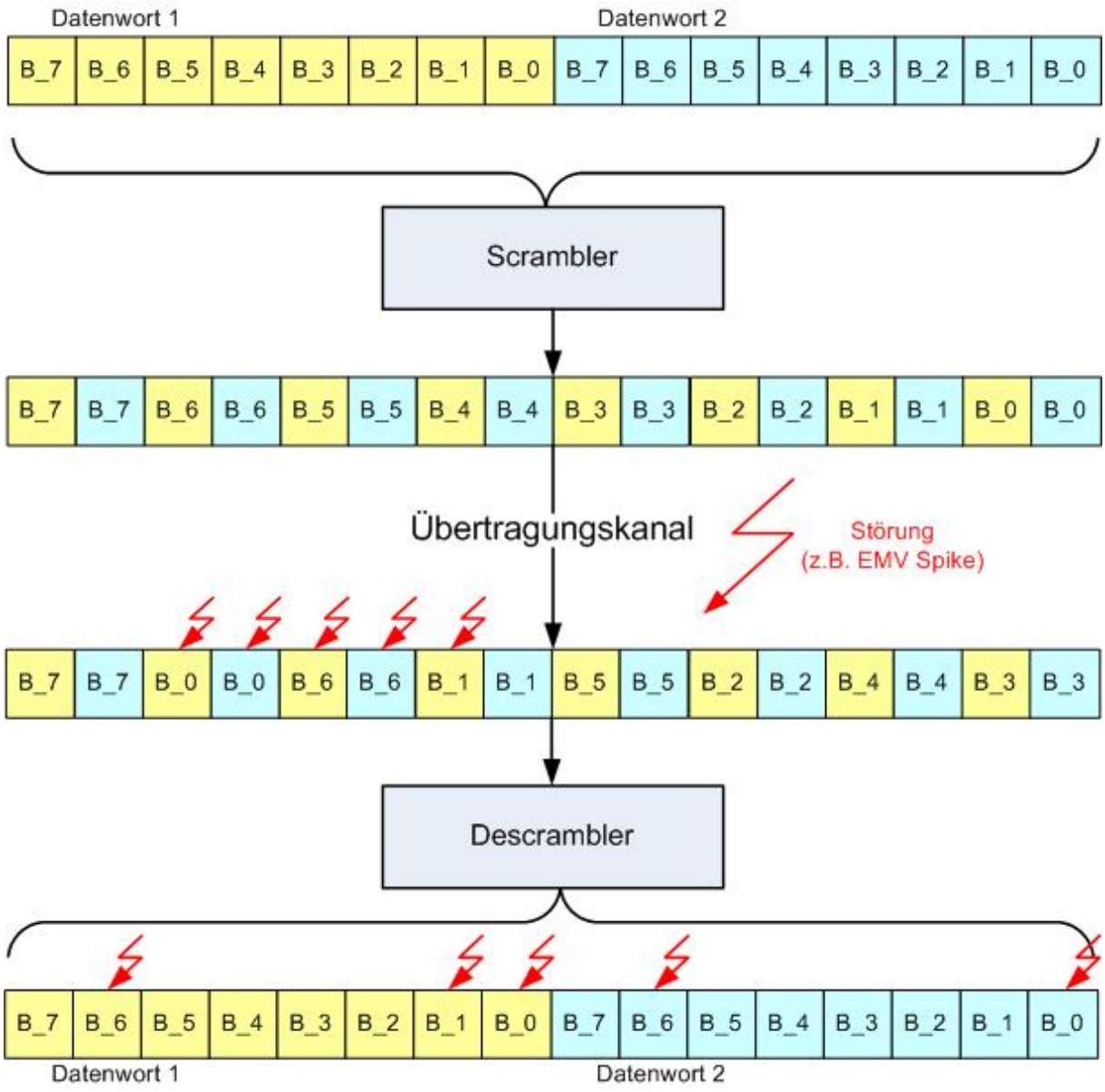

Abbildung 26: Beispiel für den Einsatz eines Verwürflers (engl. Scrambler)

Der Kodierungsalgorhythmus der Balisentelegramme bei ETCS basiert auf einem weitaus komplizierterem Algorhythmus, der in Kapitel 4.1. beschrieben ist. Er sorgt auf Grund seiner großen Hammingdistanz und seines Verwürflers dafür, dass einzelne Bitfehlerbursts bis zu einer Länge von 75 aufeinander folgenden Bits, oder mehrere Bitfehlerbursts aus kleineren Gruppen innerhalb eines Telegramms sicher erkannt werden können.

Daher sollte auf dem DSP die Möglichkeit bestehen, folgende Bitfehlerbursts in ein Telegramm einzubringen:

Tabelle 22: Einstellmöglichkeiten zur Simulation von Bitfehlerbursts

Nr.	Beschreibung
1	Einfügen eines Bursts der Länge 10 (High)
2	Einfügen eines Bursts der Länge 10 (Low)
3	Einfügen eines Bursts der Länge 52 (High)
4	Einfügen eines Bursts der Länge 52(Low)
5	Einfügen eines Bursts der Länge 75 (High)
6	Einfügen eines Bursts der Länge 75 (Low)
7	Einfügen zweier Bursts der Länge 20 (High) und 33 (High)
8	Einfügen zweier Bursts der Länge 20 (Low) und 33 (Low)
9	Einfügen zweier Bursts der Länge 20 (High) und 33 (Low)
10	Einfügen zweier Bursts der Länge 20 (Low) und 33 (High)

5.2.3 Bit Slips and Bit Insertion

Bei diesen Fehlern handelt es sich um zusätzlich eingebrachte, oder nicht erkannte Bits in einem Datenbitstrom. Dabei wird bei einem Bitslip (engl. Slip = schlüpfen) ein Bit vom Sender ausgelassen, oder vom Empfänger nicht registriert und fehlt deshalb im empfangenen Telegramm.

Bei einer Bit Insertion (engl. insert = einbringen) wird ein einzelnes Bit entweder vom Sender doppelt gesendet, oder vom Empfänger doppelt abgetastet und ausgewertet. Es ist also im Telegramm ein zusätzliches Bit vorhanden. Abbildung 27 veranschaulicht diesen Vorgang.

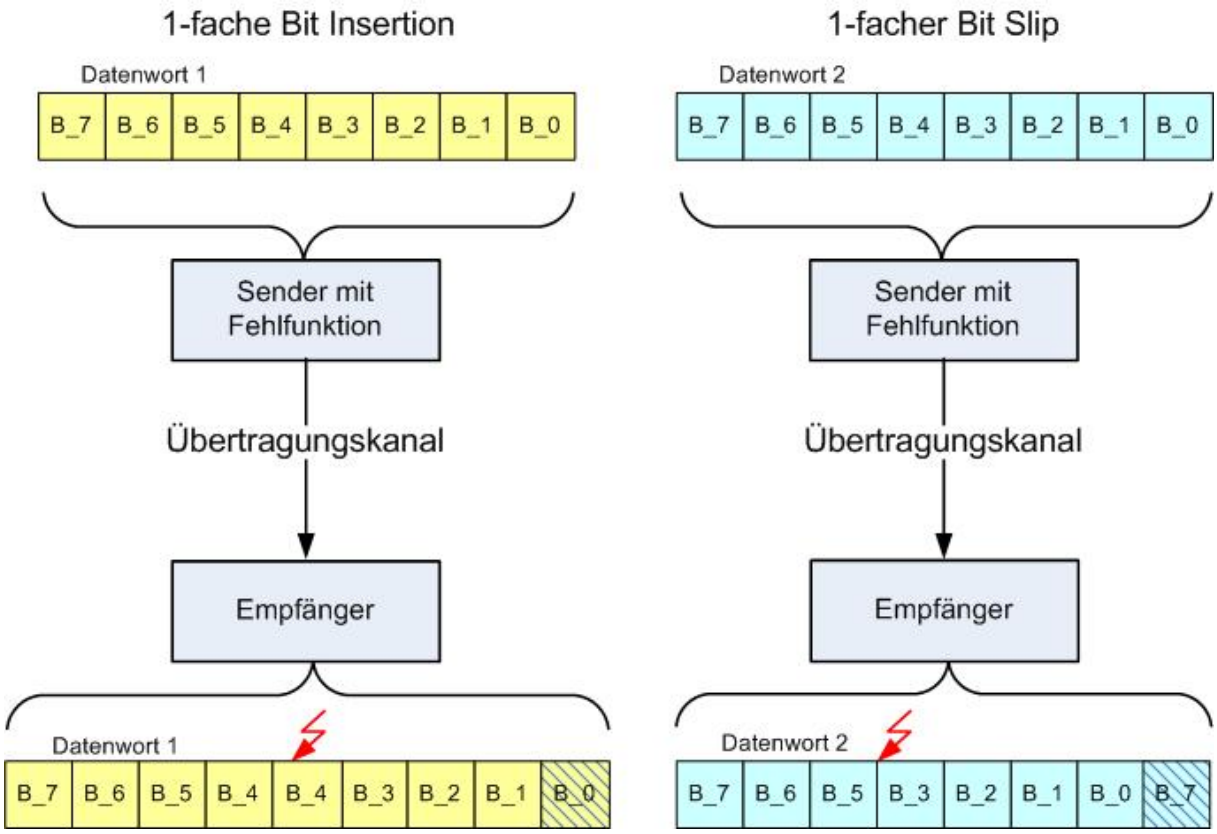

Abbildung 27: Beispiele für Bit Insertion und Bit Slip

Der bei der Kodierung der Balisentelegramme unter ETCS verwendete Algorhythmus soll die Erkennung von drei oder weniger solcher Vorkommnisse innerhalb eines Telegrammsfensters T_{Window} sicher erkennen.

Um die Länge der kodierten Telegramme nicht zu verändern kann allerdings immer nur ein Paar von Bitslip und Bitinsertion simuliert werden. Daher sollte auf dem DSP die Möglichkeit bestehen, folgende Fehler in ein Telegramm zu kodieren:

Tabelle 23: Einstellmöglichkeiten zur Simulation von Bitfehlerbursts

Nr.	Beschreibung
1	Einfügen von einer Insertion und einem Slip
2	Einfügen von zwei Insertions und zwei Slips
3	Einfügen von drei Insertions und drei Slips
4	Einfügen von vier Insertions und vier Slips

5.2.4 Unter- und Überabtastung

Jeder digitale Übertragungskanal überträgt die digitalen Daten mit einer festgelegten Datenrate (engl. Baudrate). Diese Datenrate legt fest, wie viele Datenbits innerhalb einer definierten Zeitdauer über den Datenkanal übertragen werden können.

$$< Datenrate >= \frac{< Anzahl\_der\_Bits >}{< Dauer\_der\_Übertragung >} \qquad (8)$$

$$R = \frac{N}{t} \cdot \frac{Bit}{s}$$

Formel 8: Definition der Datenrate R

Für die Übertragung digitaler Daten ist es essentiell wichtig, dass die Datenrate R bei Sender und Empfänger identisch ist. Sollte das nicht der Fall sein, so kommt es zu Übertragungsfehlern auf Grund von Überabtastung (engl. Oversampling) oder Unterabtastung (eng. Undersampling) am Empfänger.

Dieser würde z.B. bei zweifacher Überabtastung jedes Datenbit, das der Sender über den Datenkanal versendet, zweifach auswerten. Bei Unterabtastung würde der Empfänger je nach Grad der Unterabtastung nur jedes x-te Bit, das der Sender überträgt, auswerten.

Dabei muss in diesem Fall die Unterabtastung nach zwei Fällen unterschieden werden.

1. Unterabtastung durch zu hohe Datenrate am Sender
2. Unterabtastung durch weglassen jedes x-ten Bits im Telegramm am Sender

Da der erste Fall nur eine Erhöhung der Datenrate am Sender verlangt, ist dieser Fehlerfall, genau wie die Überabtastung sehr einfach nachzustellen. Der zweite Fall erfordert allerdings eine Neukodierung des Telegramms bei gleich bleibender Datenrate. Die Umsetzung dieses Falls ist also nicht trivial zu lösen. Ein Beispiel für Bitfolgen mit Unter- und Überabtastung zeigt Abbildung 28.

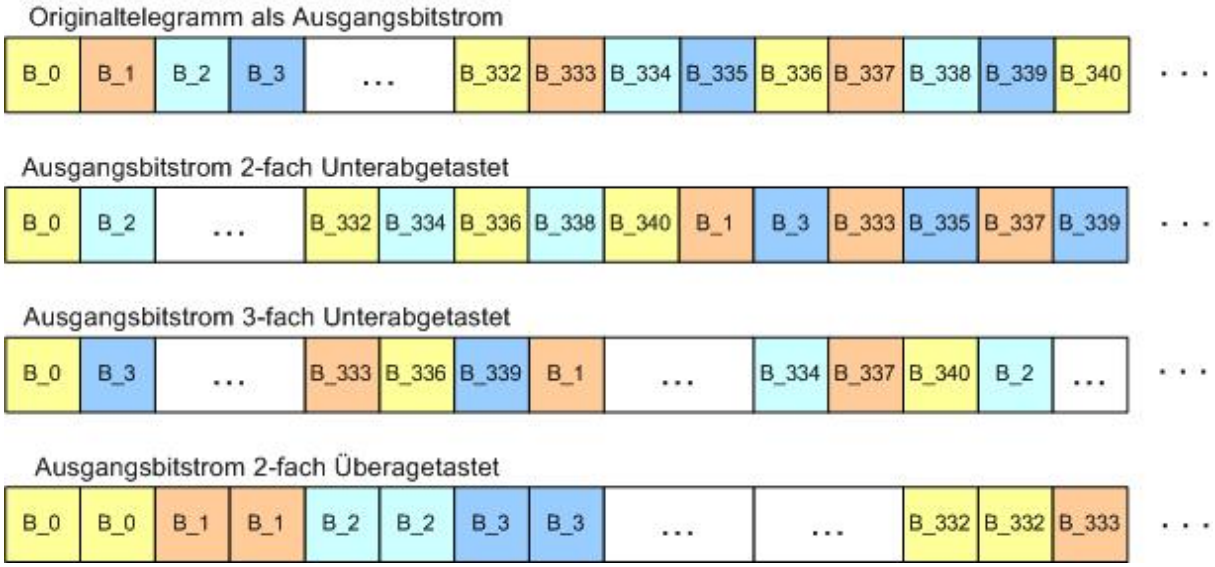

Abbildung 28: Beispiel Unter- und Überabgetastete Telegramme

Unter- und Überabtastung sind ein Problem, da in vielen zyklischen Kodes eine Unter- oder Überabtastung mit einem Grad von 2^k wieder zu gültigen Kodeworten führt. Der für die Kodierung der Balisentelegramme in ETCS verwendete Kode soll die Erkennung folgender Über- und Unterabtastungen erkennen.

- ⇨ Überabtastung von ganzzahligen Graden kleiner als 8
- ⇨ Überabtastung von allen Graden größer als 8
- ⇨ Unterabtastung für die Grade 2, 4, 8 und 16

Deshalb sollen folgende Arten der Unter- und Überabtastung auf dem DSP simuliert werden:

Tabelle 24: Einstellmöglichkeiten zur Simulation von Unter- und Überabtastung

Nr.	Beschreibung
1	Überabtastung mit Grad 2
2	Überabtastung mit Grad 3 (nicht feststellbar)
3	Überabtastung mit Grad 4
4	Überabtastung mit Grad 8
5	Überabtastung mit Grad 9
6	Unterabtastung mit Grad 2
7	Unterabtastung mit Grad 3 (nicht feststellbar)
8	Unterabtastung mit Grad 4
9	Unterabtastung mit Grad 8
10	Unterabtastung mit Grad 16

5.3 Umsetzung der Fehlermodelle auf dem DSP

Für die Simulation von Übertragungsfehlern auf dem DSP wird ein neues RS 232 Kommando implementiert. Über den Befehl **RS2DSP_SetTxErrorState** kann im DSP ein Fehlermodell aktiviert werden. Anschließend werden alle Balisentelegramme mit diesem Fehlermodell über die HF - Schnittstelle übertragen. Die Fehlermodelle entsprechen dabei den Definitionen im Kapitel 5.2. Als Parameter müssen der Funktion folgende Werte übergeben werden:

Tabelle 25: Parameter RS 232 Kommandos RS2DSP_SetTxErrorState (…)

Name	Typ	Beschreibung
nState	Uint32	Nummer des Fehlermodells, das aktiviert werden soll
para	Uint32	Freier Parameter, der der Angabe von Zusatzinformationen dient.

Dabei soll die Nummer des Fehlermodells grob der Einteilung in die verschiedenen Fehlermodelle aus Kapitel 5.2 entsprechen. Im freien Parameter wird dann festgelegt, welches genaue Fehlermodell aktiviert werden soll. So kann z.B. zur Aktivierung eines Einzelbitfehlers in nState „Einzelbitfehler“ und in para „1“ eingetragen werden.

Bei einem anschließenden Starten einer Sequenz oder Senden eines Telegramms wird das aktivierte Fehlermodell in jedes zu sendende Telegramm eingebracht. Zu diesem Zweck wird eine neue Funktion **prepareTxError (…)** implementiert. Diese Funktion wird aus der bereits vorhandenen Funktion **prepareTlgForTx (…)** aufgerufen und bringt je nach Fehlermodell Fehler in die zu sendenden Telegramme ein, bevor diese zum Versenden vervielfacht werden. Das Zusammenspiel dieser beiden Funktionen beschreibt Abbildung 29.

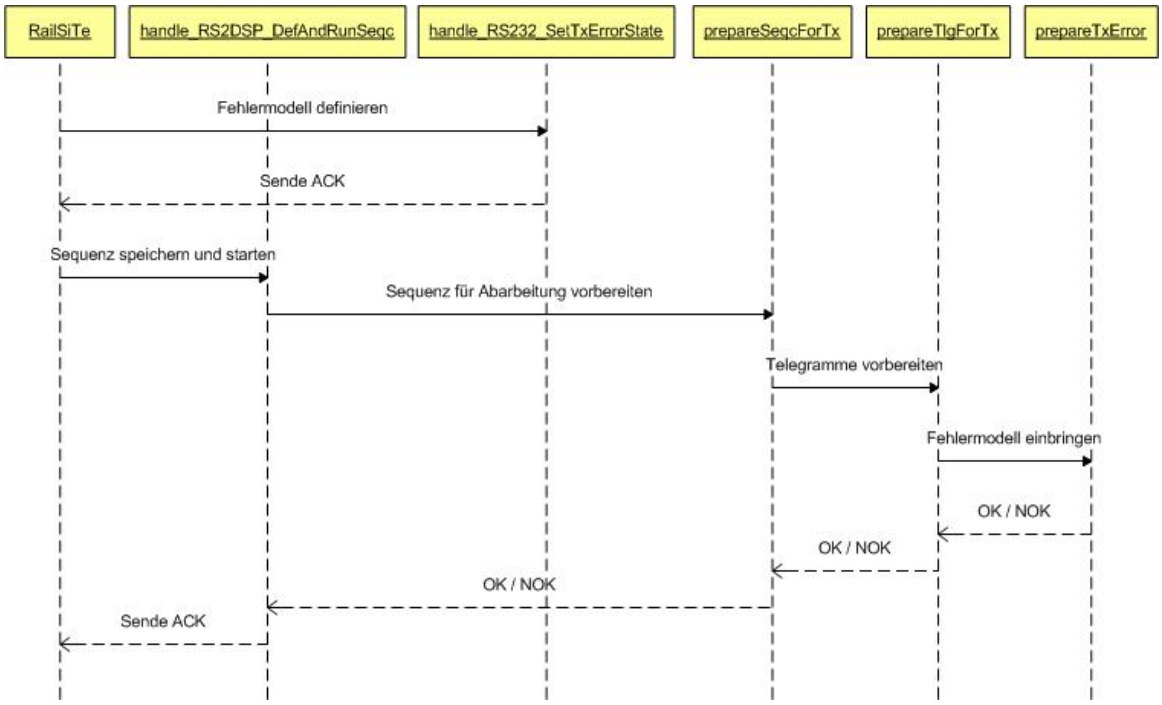

Abbildung 29: Sequenzdiagramm zum Einstellen von TX Fehlermodellen

Da die Funktion **prepareTxError (…)** nicht direkt aus der Funktion **handle_RS2DSP_SetTxErrorState (…)** aufgerufen wird, müssen alle notwendigen Daten in einer globalen Struktur gespeichert werden, damit sie beim Aufruf der Funktion **prepareTxError (…)** zur Verfügung stehen. Die Struktur hat folgenden Aufbau:

```
Typedef struct
{
        Uint32 txErr_State_u32;          // Nummer des zu aktivierenden Fehlermodells
        Uint32 txErr_param_u32;          // zusätzlicher Parameter
        uint8  txErr_valid_states_aru8[TX_ERROR_NUMBER];    // Liste aller gült. Modelle
} t_txError_status
```

Zur Simulation der einzelnen Fehlermodelle wird die Funktion **prepareTxError (…)** in weitere Unterfunktionen unterteilt, die jeweils eines der Fehlermodelle aus Kapitel 5.2 simulieren können. Diese sind:

- ⇨ **prepareRandBitError (…):**

  Diese Funktion erzeugt in einem Telegramm Einzelbitfehler an zufälligen Positionen. Die Anzahl der Fehler wird dabei als Parameter übergeben.

- ⇨ **prepareBitBurstError (…):**

  Diese Funktion erzeugt in einem Telegramm zufällige Bitfehlerbursts. Die Anzahl, Länge und Polarität werden dabei als Parameter übergeben.

- ⇨ **prepareBitSlipError (…):**

  Diese Funktion erzeugt zusätzliche Bits, oder fehlende Bits im Telegramm. Um die Telegrammlänge beizubehalten, kann immer nur die Kombination von einem fehlenden und einem zusätzlichen Bit simuliert werden.

- ⇨ **prepareOverSmplError (…):**

  Diese Funktion erzeugt eine Überabtastung des Telegrams. Da diese Funktion leider nicht durch das einfache Ändern der Übertragungsfrequenz möglich ist (Teiler für McBSP Taktrate ist zu klein), wird das Telegramm bei Aufruf dieser Funktion um den Überabtastungsfaktor vervielfacht.

⇨ **prepareUnderSmplError (…):**

Diese Funktion erzeugt eine Unterabtastung des Telegrams. Die Telegrammlänge bleibt dabei erhalten.

Grundsätzlich muss davon ausgegangen werden, dass die Telegramminhalte durch das Einbringen der Fehlermodelle geändert werden. Um zu verhindern, dass alle Telegramme, die mit einem Fehlermodell versendet wurden, anschließend neu kodiert werden müssen, wird die Funktion **prepareTlgForTx (…)** so geändert, dass das komplette Telegramm vor Aufruf der Funktion **prepareTxError (…)** in einen lokalen Buffer kopiert und nur dort geändert werden.

Um Rechenzeit zu sparen, kann immer nur ein Fehlermodell aktiv sein.

Alle Unterfunktionen, die in der Funktion **prepareTxError (…)** benutzt werden, werden im nachfolgenden näher erläutert.

5.3.1 Funktion prepareRandBitError (…)

Diese Funktion soll in ein gegebenes Telegramm Einzelbitfehler an zufälligen Positionen einfügen. Zu diesem Zweck werden als Parameter ein Zeiger auf den eigentlichen Telegramminhalt, sowie die Anzahl der Einzelbitfehler und das Telegrammformat übergeben.

Als Rückgabewert wird ein integer Wert zurückgegeben, in dem gemeldet wird, ob die Funktion fehlerfrei ausgeführt wurde. Zusätzlich wird in der Variable **numbits_u16** die Länge des resultierenden Telegramms in Bit nach der Einbringung der Fehler gespeichert. Diese bleibt allerdings bei der Einbringung von Einzelbitfehlern immer unverändert. Alle Parameter und Rückgabewerte sind in den folgenden Tabellen aufgeführt.

Tabelle 26: Parameter der Funktion prepareRandBitError (…)

Name	Typ	Beschreibung
Tlg_buffer_pu8	Uint8*	Zeiger auf das Bitfeld mit den Telegramminhalten
Tlg_format_u8	Uint8	Format des Telegramms, in das die Fehler eingefügt werden sollen: TLG_TYPE_SHORT: kurzes Telegramm TLG_TYPE_LONG: langes Telegramm
Err_number_u16	Uint16	Anzahl der einzubringenden Einzelbitfehler
numbits_u16	uint16*	Pointer auf 16Bit Variable die nach Ende der Funktion die Länge des resultierenden Telegrams in Bit enthält.

Tabelle 27: Rückgabewert der Funktion prepareRandBitError (…)

Name	Typ	Beschreibung
Return	Int	Fehlercode: ERR_OK: kein Fehler …

Zu Beginn der Funktion wird der Telegrammtyp bestimmt und alle benötigten Variablen initialisiert. Dazu gehören auch die Bitpositionen, an denen Einzelbitfehler eingefügt werden sollen. Sie werden per Zufallsgenerator erzeugt und in einem Array **temp_err_pos_aru16** gespeichert.

Nach der Erzeugung der Fehlerpositionen müssen diese nach ihrer Abfolge im Telegramm (aufsteigende Bitposition) sortiert werden, um bei der nachfolgenden Erzeugung der Einzelbitfehler das Originaltelegramm nur einmal durchlaufen zu müssen.

Anschließend werden in einer Schleife alle Bits an den vorher ermittelten Positionen in ihrem Wert gedreht. Die Bitfehler werden erzeugt. Eine Übersicht über die Funktion zeigt Abbildung 30.

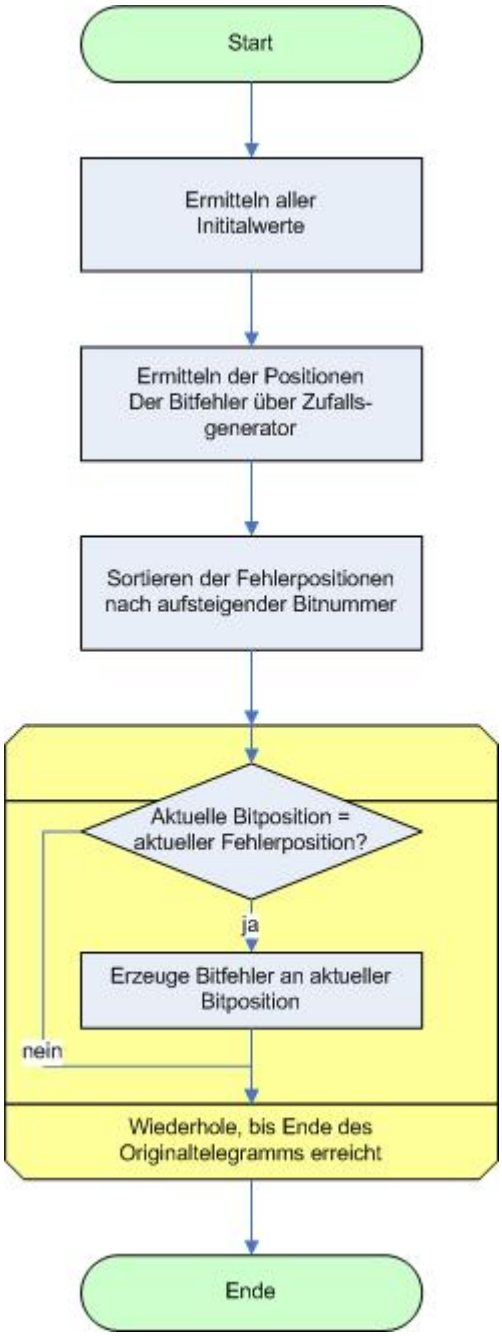

Abbildung 30: Flussdiagramm prepareRandBitError (…)

5.3.2 Funktion prepareBitBurstError (…)

Diese Funktion soll in ein gegebenes Telegramm Bitfehlerbursts einbringen. Dazu muss als Parameter in der Variable **burst_mode_u32** die Nummer des zu aktivierenden Bitburstfehlermodells übergeben werden. Anschließend bestimmt die Funktion selbsttätig die zufälligen Startpositionen der Fehlerbursts und fügt sie ein.

Als Rückgabewert wird ein integer Wert zurückgegeben, in dem gemeldet wird, ob die Funktion fehlerfrei ausgeführt wurde. Zusätzlich wird in der Variable **numbits_u16** die Länge des resultierenden Telegramms in Bit nach der Einbringung der Fehler gespeichert. Alle Parameter und Rückgabewerte sind in den folgenden Tabellen aufgeführt.

Tabelle 28: Parameter der Funktion prepareBitBurstError (…)

Name	Typ	Beschreibung
Tlg_buffer_pu8	Uint8*	Zeiger auf das Bitfeld mit den Telegramminhalten
Tlg_format_u8	Uint8	Format des Telegramms, in das die Fehler eingefügt werden sollen: TLG_TYPE_SHORT: kurzes Telegramm TLG_TYPE_LONG: langes Telegramm
burst_mode_u32	Uint16	Nummer des Bitburstfehlermodells.
numbits_u16	uint16*	Pointer auf 16Bit Variable die nach Ende der Funktion die Länge des resultierenden Telegrams in Bit enthält.

Tabelle 29: Rückgabewert der Funktion prepareBitBurstError (…)

Name	Typ	Beschreibung
Return	Int	Fehlercode: ERR_OK: kein Fehler …

Zu Beginn der Funktion wird der Telegrammtyp bestimmt und alle benötigten Variablen initialisiert. Dazu gehören auch die Bitpositionen, an denen die Bitfehlerbursts eingefügt werden sollen. Die Anzahl und Länge der Bursts ist durch das ausgewählte Bitburstfehlermodell festgelegt.

Dabei dient die Auswahl des Nutzers in der Variable **burst_mode_u32** als Index für die Struktur **txErr_burstmode_s** in dem alle Bitburstfehlermodelle definiert sind. Grundsätzlich können nur zwei Bitfehlerbursts mit beliebiger Länge und Polarität in ein Telegramm eingefügt werden. Die Struktur **txErr_burstmode_s** hat folgenden Aufbau:

```
typedef struct
{
        uint8 burst1_len_u8;    // length of burst 1
        uint8 burst2_len_u8;    // length of burst 2
        uint8 burst1_pol_u8;    // polarity of burst 1
        uint8 burst2_pol_u8;    // polarity of burst 2
} t_txErr_burstmode;
```

Mit den Daten aus der Struktur **txErr_burstmode_s** können die Startpositionen der einzelnen Bursts bestimmt werden. Um zu verhindern, dass die Bursts die Telegrammgrenzen verletzen, muss zur Berechnung der Startpositionen folgende Formel benutzt werden.

$$Startposition = RandomValue(x) \quad \textbf{(9)}$$

mit $x \in N \cap \{0...(< Telegrammlänge\_in\_Bit > - < Burstlänge\_in\_Bit > -1)\}$

Formel 9: zulässiger Wertebereich für Burststartposition

Zu diesem Zweck wurde eine neue Funktion **getRandVal (…)** implementiert, die zufällige, ganzzahlige Werte aus einem definierten Wertebereich mit gleicher Auftretenswahrscheinlichkeit erzeugt. Der Funktion wird die obere Grenze des zulässigen Wertebereiches als Parameter übergeben. Der Rückgabewert enthält dann die erzeugte Zufallszahl.

Nachdem die Startpositionen der Bursts gefunden wurden, werden die Fehler in das Telegramm eingebracht. Eine Übersicht über den Ablauf der Funktion zeigt Abbildung 31.

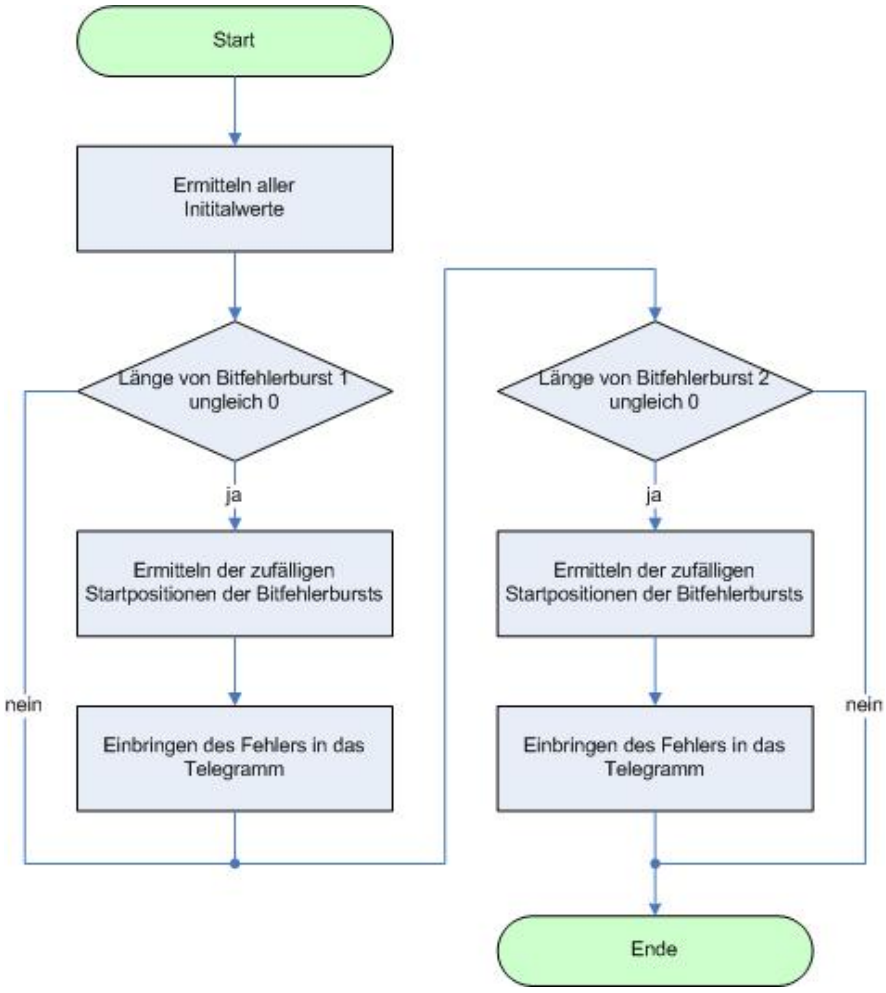

Abbildung 31: Flussdiagramm prepareBitBurstError (…)

5.3.3 Funktion prepareBitSlipError (…)

Diese Funktion soll in ein gegebenes Telegramm Bitslips (engl. slip = schlüpfen, hier auslassen eines Bits aus dem Telegramm) und Bitinsertions (engl. insertion = einfügen, hier einfügen eines zusätzlichen Bits in das Telegramm) einbringen. Dazu muss als Parameter das Bitslipfehlermodell in der Variable **slip_mode_u32** an die Funktion übergeben werden. Sie beschreibt die Anzahl der Bitlslips und Bitinsertions, die durchgeführt werden sollen. Da die Telegrammlänge durch diese Funktion nicht geändert werden darf, wird immer ein Paar aus Bitslip und Bitinsertion generiert.

Als Rückgabewert wird ein integer Wert zurückgegeben, in dem gemeldet wird, ob die Funktion fehlerfrei ausgeführt wurde. Zusätzlich wird in die Variable **numbits_u16** die Länge des resultierenden Telegramms in Bit gespeichert. Alle Parameter und Rückgabewerte sind in den folgenden Tabellen aufgeführt.

Tabelle 30: Parameter der Funktion prepareBitSlipError (…)

Name	Typ	Beschreibung
Tlg_buffer_pu8	Uint8*	Zeiger auf das Bitfeld mit den Telegramminhalten
Tlg_format_u8	Uint8	Format des Telegramms, in das die Fehler eingefügt werden sollen: TLG_TYPE_SHORT: kurzes Telegramm TLG_TYPE_LONG: langes Telegramm
slip_mode_u32	Uint16	Anzahl der Bitslip / Bitinsertion Paare
numbits_u16	uint16*	Pointer auf 16Bit Variable die nach Ende der Funktion die Länge des resultierenden Telegrams in Bit enthält.

Tabelle 31: Rückgabewert der Funktion prepareBitSlipError (…)

Name	Typ	Beschreibung
Return	Int	Fehlercode: ERR_OK: kein Fehler …

Zu Beginn der Funktion wird der Telegrammtyp bestimmt und alle benötigten Variablen initialisiert. Anschließend werden die zufälligen Bitpositionen der Fehler im Telegramm erzeugt. Dabei muss darauf geachtet werden, dass keine zwei Fehler an der gleichen Bitposition liegen, da sonst ein mehrfacher Durchlauf durch das Telegramm bei Einbringen der Fehler notwendig wäre. Sollten zwei Fehler die gleiche Position haben, so werden alle Fehlerpositionen neu generiert.

Ist die Erzeugung der Fehlerpositionen erfolgreich, so werden die einzelnen Fehler in das Telegramm eingebracht. Dabei wird immer nach einem Bitslip eine Bitinsertion durchgeführt, so dass das Telegramm nach Einbringen zweier Fehler die Länge des Ausgangstelegramms besitzt. Beim Einbringen eines Bit wird das Originalbit an der Fehlerposition dupliziert.

Eine Übersicht über den Ablauf der Funktion zeigt Abbildung 32.

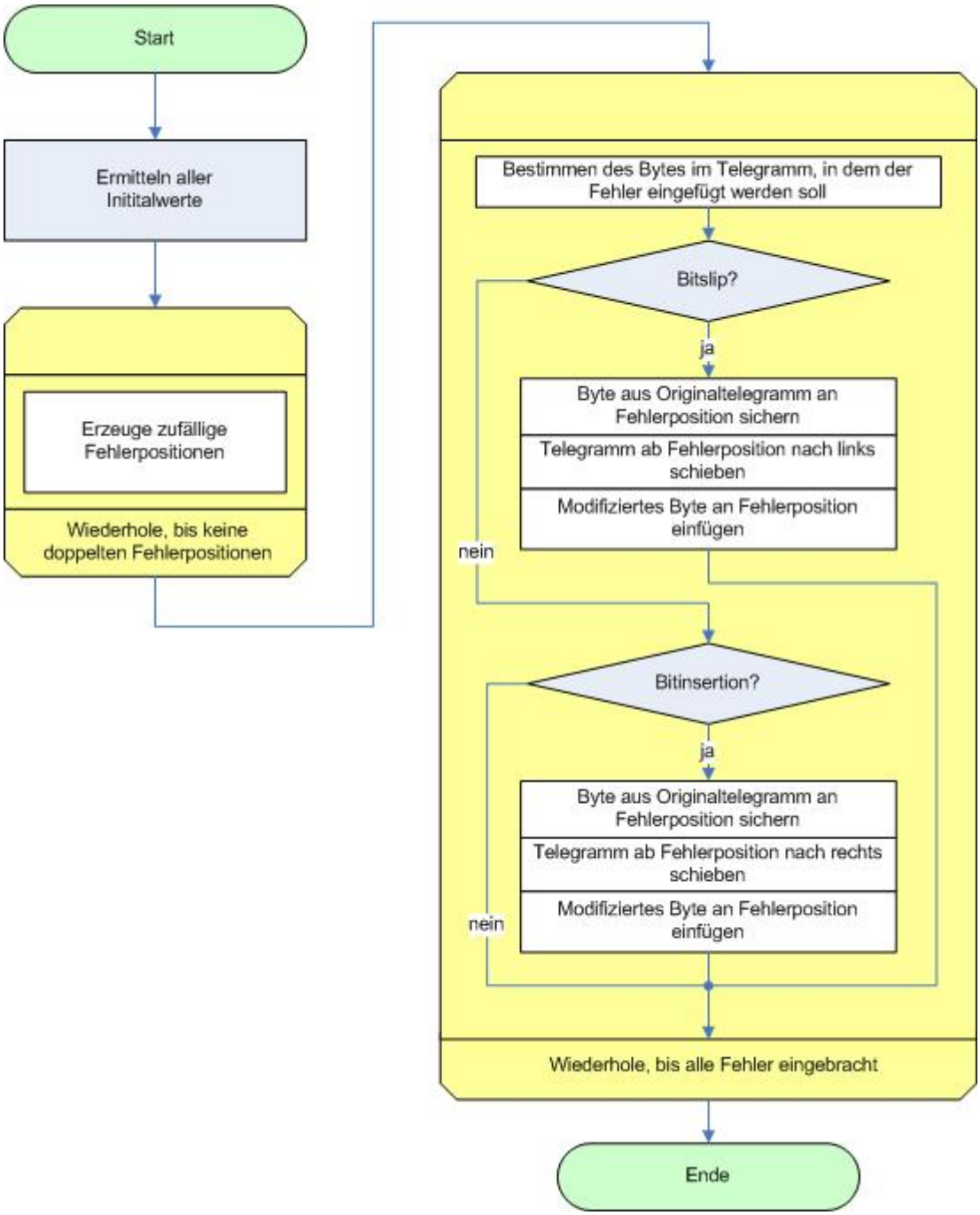

Abbildung 32: Flussdiagramm prepareBitSlipError (…)

Wichtig:

Durch die Überprüfung auf doppelte Fehlerpositionen kann diese Funktion theoretisch unendlich Lange dauern, wenn immer doppelte Fehlerpositionen erzeugt werden. Die Wahrscheinlichkeit hierfür ist in unserer Anwendung gering. Sie steigt jedoch mit der Anzahl der Bitslip / Bitinsertion Paare.

Wenn angenommen wird, dass alle Ereignisse des Zufallszahlengenerators gleich wahrscheinlich sind, dann berechnet sich die Wahrscheinlichkeit einer doppelten Fehlerposition bei einem Durchlauf wie folgt.

Wahrscheinlichkeit eines Ereignisses p_{Einzel}:

$$p_{Einzel} = \frac{1}{< Anzahl\_Bits\_im\_Telegramm >}$$

$$p_{Einzel\_short} = \frac{1}{341} = 0{,}002933$$

$$p_{Einzel\_long} = \frac{1}{1023} = 0{,}000978$$

Wahrscheinlichkeit einer doppelten Position p_{Doppel}:

$$p_{Doppel} = P\left(p_{Einzel} \middle| p_{Einzel}\right) = \frac{p_{Einzel} \cdot p_{Einzel}}{p_{Einzel}} = p_{Einzel}$$

Bei mehreren Bitslip / Bitinsertion Paaren erhöht sich diese Wahrscheinlichkeit allerdings mit der Anzahl der Durchläufe. Die Wahrscheinlichkeit, dass eine doppelte Fehlerposition in der Funktion erzeugt wird ist also:

$$p_{Doppel} = N \cdot p_{Einzel} \qquad \textbf{(10)}$$

mit $N = slip\_\mathrm{mod}e\_u32$

Formel 10: Wahrscheinlichkeit für doppelte Fehlerpositionen in prepareBitSlipError (…)

Diese Werte gelten allerdings nur näherungsweise, da es sich bei dem Zufallsgenerator im DSP nur um einen Pseudozufallsgenerator handelt, der nach einem Neustart der Software immer die gleichen Sequenzen von Zufallswerten erzeugt.

5.3.4 Funktion prepareOverSmplError (…)

Diese Funktion soll ein x-faches Überabtasten des Originaltelegramminhalts ermöglichen. Dabei wird jedes Bit der Originalbitfolge x-fach vervielfacht. Der Überabtastungsfaktor wird dabei als Parameter in **sample_mode_u32** übergeben.

Als Rückgabewert wird ein integer Wert zurückgegeben, in dem gemeldet wird, ob die Funktion fehlerfrei ausgeführt wurde. Zusätzlich wird in die Variable **numbits_u16** die Länge des resultierenden Telegramms in Bit geschrieben. Die Länge ändert sich je nach Wert des Überabtastungsfaktors. Alle Parameter und Rückgabewerte sind in den folgenden Tabellen aufgeführt.

Tabelle 32: Parameter der Funktion prepareOverSmplError (…)

Name	Typ	Beschreibung
Tlg_buffer_pu8	Uint8*	Zeiger auf das Bitfeld mit den Telegramminhalten
Tlg_format_u8	Uint8	Format des Telegramms, in das die Fehler eingefügt werden sollen: TLG_TYPE_SHORT: kurzes Telegramm TLG_TYPE_LONG: langes Telegramm
sample_mode_u32	Uint16	Anzahl der Bitslip / Bitinsertion Paare
numbits_u16	uint16*	Pointer auf 16Bit Variable die nach Ende der Funktion die Länge des resultierenden Telegrams in Bit enthält.

Tabelle 33: Rückgabewert der Funktion prepareOverSmplError (…)

Name	Typ	Beschreibung
Return	Int	Fehlercode: ERR_OK: kein Fehler …

Für die Überabtastung wird eine Aufteilung in verschiedene Funktionen angedacht. Es wird eine zusätzliche Funktion **doXfoldOverSmpl (…)** implementiert, die eine Überabtastung mit einem Überabtastungsfaktor von null bis acht durchführen kann.

Soll eine höhere Überabtastung erreicht werden, so muss diese Funktion innerhalb der Funktion **prepareOverSmplErr (…)** mehrfach aufgerufen werden. Dabei muss jeweils der aktuelle Telegramminhalt mit der aktuellen Telegrammlänge als Parameter übergeben werden. Für eine neunfache Überabtastung muss also zweimal die Funktion **doXfoldOverSmpl (…)** mit einem Überabtastungsfaktor von drei aufgerufen werden. In der jetzigen Version der DSP Software soll nur eine Überabtastung bis zu einem Faktor von neun realisiert werden, da größere Werte zu zulangen Folgen von logisch eins oder logisch null führen, um gültige Kodeworte zu erzeugen.

Eine Übersicht über die Funktion **prepareOverSmplErr (…)** bietet Abbildung 33.

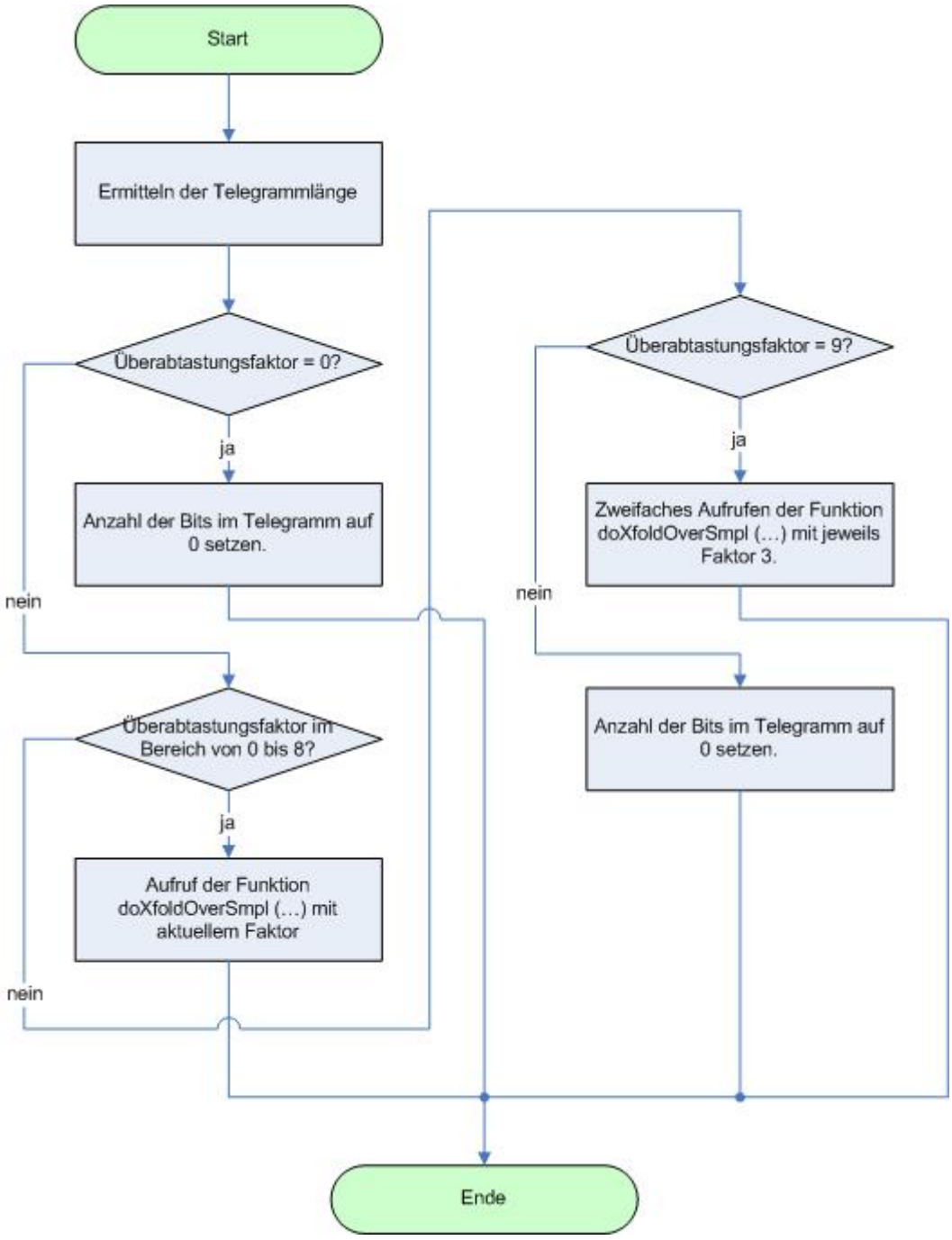

Abbildung 33: Flussdiagramm prepareOverSmplError (…)

Wichtig:

Theoretisch können mit diesem Aufbau der Überabtastung beliebig große Überabtastungsfaktoren erreicht werden. Allerdings wird in der Funktion **prepareOverSmplError (…)** keine Überprüfung der Array Grenzen des Telegrammpuffers durchgeführt. Die aufrufende Funktion muss also selbst dafür sorgen, dass der Telegrammspeicherplatz ausreichend groß bemessen ist.

5.3.5 Funktion prepareUnderSmplError (…)

Diese Funktion soll ein X-faches Unterabtasten (engl. Undersampling) ermöglichen. Sie bildet damit das direkte Gegenteil zu **prepareOverSmplError (…)**, indem nur jedes X-te Bit der Originalbitfolge übertragen wird. Dazu muss das Originaltelegramm vor dem versenden neu zusammengesetzt werden. Der Faktor der Unterabtastung (X) wird als Parameter in der Variable **sample_mode_u32** übergeben.

Als Rückgabewert wird ein integer Wert zurückgegeben, in dem gemeldet wird, ob die Funktion fehlerfrei ausgeführt wurde. Zusätzlich wird in die Variable **numbits_u16** die Länge des resultierenden Telegramms in Bit geschrieben. Alle Parameter und Rückgabewerte sind in den folgenden Tabellen aufgeführt.

Tabelle 34: Parameter der Funktion prepareUnderSmplError (…)

Name	Typ	Beschreibung
Tlg_buffer_pu8	Uint8*	Zeiger auf das Bitfeld mit den Telegramminhalten
Tlg_format_u8	Uint8	Format des Telegramms, in das die Fehler eingefügt werden sollen: TLG_TYPE_SHORT: kurzes Telegramm TLG_TYPE_LONG: langes Telegramm
sample_mode_u32	Uint16	Anzahl der Bitslip / Bitinsertion Paare
numbits_u16	uint16*	Pointer auf 16 Bit Variable die nach Ende der Funktion die Länge des resultierenden Telegrams in Bit enthält.

Tabelle 35: Rückgabewert der Funktion prepareUnderSmplError (…)

Name	Typ	Beschreibung
Return	Int	Fehlercode: ERR_OK: kein Fehler …

Die Unterabtastung wird durch ein Umsortieren des Originaltelegramms erreicht. Zu diesem Zweck wird eine globale Struktur **txErr_undersmpl_s** angelegt. Sie enthält eine Menge von Unterabtastungsfaktor vielen Einträgen folgender Struktur:

```
typedef struct
{
        uint8  data_aru8 [TLG_LONG_ENCODED_BYTE_SIZE]; // array for undersmpl tlg parts
        uint8  bitmask_aru8;            // bitmask for underdsampled data array
        uint16 bitcnt_aru16;            // counter for bits in array
        uint16 bytecnt_aru16;           // byte index for access on array
} t_txErr_undersmpl;
```

Diese Struktur muss global angelegt werden, da nur globale Datenstrukturen in das externe SDRAM ausgelagert werden können.

In der Funktion **prepareUnderSmplError (…)** wird in einer Schleife jedes Bit des Originaltelegramms einzeln untersucht und einem der X Arrays in der Struktur **txErr_undersmpl_s** zugeordnet, die jeweils einen Teil des Originaltelegramms speichern.

So werden z.B. bei einem Unterabtastungsfaktor von drei die Bits

- ⇨ **{0; 3; 6; 9; …}** im ersten Array,
- ⇨ **{1; 4; 7; 10; …}** im zweiten Array und schließlich die Bits
- ⇨ **{2; 5; 8; 11; …}** im dritten Array gespeichert.

Durch dieses Vorgehen ist das Originaltelegram am Ende in X einzelne Arrays aufgeteilt, die jeweils für sich einen unterabgetasteten Teil des Originaltelegramms enthalten. Anschließend werden die einzelnen Telegrammteile wieder zu einem ganzen Telegramm mit der Länge des Originaltelegramms zusammengesetzt.

Durch dieses Vorgehen bleibt die Länge des Telegramms erhalten. Eine Übersicht über die Funktion **prepareUnderSmplError (…)** bietet Abbildung 34.

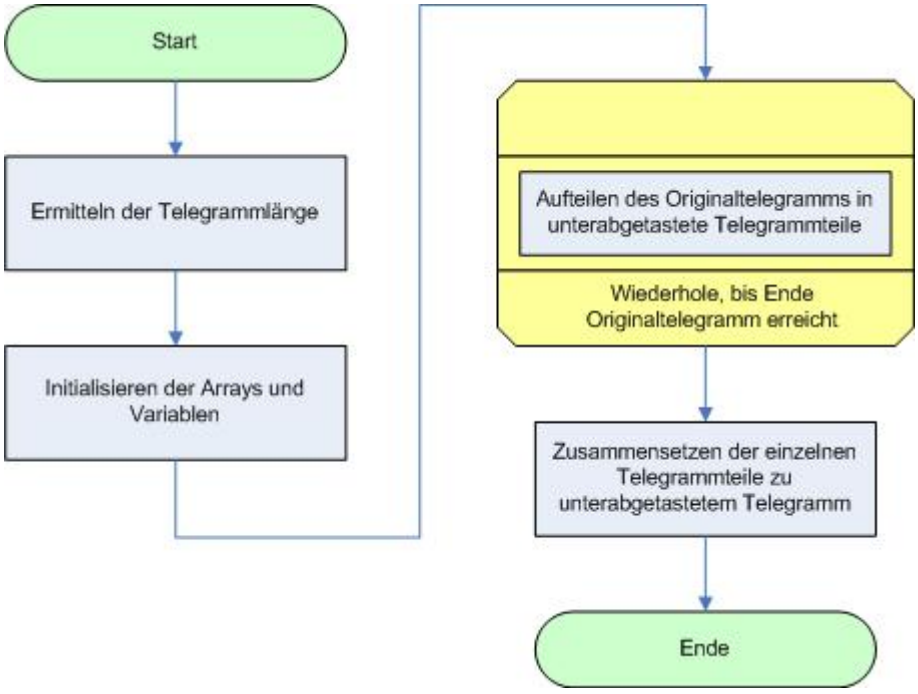

Abbildung 34: Flussdiagramm prepareUnderSmplError (…)

5.4 Performance Untersuchung TX-Fehlermodelle

Da die Software auf dem DSP in der späteren Anwendung als Balisenmodul im RailSiTe Labor in Echtzeit arbeiten soll, werden auch für die Simulation der TX - Fehlermodelle Performanceuntersuchungen angestellt.

Dadurch soll verhindert werden, dass Funktionen in der DSP - Software implementiert werden, die später durch sehr lange Rechenzeit zur Blockierung des RailSiTe - Labors führen.

Alle hier ermittelten Messergebnisse wurden mit einem Oszilloskop aufgenommen. Zu diesem Zweck wurde in der DSP - Software vor dem Start der Funktion **prepareTxError (…)** ein digitaler Ausgang des DSPs gesetzt und nach Ende der Funktion wieder gelöscht. Dadurch konnten sehr genaue Messwerte über die Dauer der einzelnen Funktionen zur Simulation der TX - Fehlermodelle ermittelt werden.

Alle Messergebnisse sind in der nachfolgenden Tabelle aufgelistet.

Tabelle 36: Performance Ergebnisse der TX – Fehlermodelle

Fehlermodell	Parameter / Dauer in µs									
Kein TX Modell aktiv	-	-	-	-	-	-	-	-	-	-
	8,0	8,0	8,0	8,0	8,0	8,0	8,0	8,0	8,0	8,0
Random Bit Error	1	..	..	..	..	..	..	..	..	20
	54,4	..	..	..	..	..	..	..	..	126,4
Bitburst Error	0	1	2	3	4	5	6	7	8	9
	12,6	16,4	16,0	16,2	17,4	13,8	20,6	15,8	19,6	20,4
Bitslip Error	0	1	2	3	4	-	-	-	-	-
	12,8	23,2	34,8	60,4	72,4	-	-	-	-	-
Oversampling Error	0	1	2	3	4	5	6	7	8	9
	12,8	194,8	195,2	258,0	200,2	368,0	392,0	524,0	210,4	980,2
Undersampling Error	0	1	2	3	4	5	6	7	8	16
	680,0	414,0	434,2	460,0	476,0	498,0	516,0	536,0	558,0	710,0

Aus der Tabelle lässt sich schließen, dass die Dauer des längsten (sinnvollen) TX - Fehlermodells ca. 1 ms beträgt (9-faches Oversampling). Daraus ergibt sich die Forderung seitens der DSP - Software an das RailSiTe - Labor, dass es bei Start einer Sequenz bei aktiviertem TX – Fehlermodell mindestens eine Wartezeit von 1ms zusätzlich einrechnet.

Das ist aber kein Problem, da diese Zeit von der Pausenzeit des ersten Telegramms der zu versendenden Sequenz abgezogen werden kann und mit maximal 1ms immer noch sehr gering ausfällt.

Die zusätzliche Zeit, die durch die TX – Fehlermodell Funktionalität bei keinem aktivierten Modell entsteht ist mit 8µs zu vernachlässigen.

Im Anhang sind Oszilloskopbilder der unmodulierten Balisentelegramme mit implementiertem Fehlermodell zu finden.

6 Erweiterung der Diagnoseschnittstelle

6.1 Anforderungen

Die Diagnoseschnittstelle soll es dem Anwender ermöglichen, den DSP in einen Diagnose Modus zu versetzen. In diesem Betriebsmodus sollen alle RS 232 Pakete, die das RailSiTe an den DSP sendet, ignoriert werden. Der DSP reagiert dann ausschließlich auf Anfragen über den Diagnosekanal, bis er wieder in den normalen Betriebsmodus zurückversetzt wird.

Wenn der DSP in den Diagnose Modus versetzt wurde, können verschiedene Kommandos auf dem DSP ausgeführt werden, die zu allgemeinen Diagnosezwecken benutzt werden. So soll es z.B. möglich sein, ein Telegramm unendlich lange zyklisch zu senden, oder einen Performance Test der Kodierung zu starten.

Eine Übersicht über alle Funktionen, die im Diagnose Modus auf dem DSP ausgeführt werden können sollen, ist in der nachfolgenden Tabelle gegeben.

Tabelle 37: Übersicht Funktionalität im Diagnosemodus

Nr.	Schlagwort	Beschreibung
1	Aktivierung Diagnosemodus	Es soll möglich sein, den Diagnose Modus über ein RS 232 Kommando zu aktivieren und zu deaktivieren.
2	Loglevel setzen	Es soll möglich sein, den Loglevel für alle Debugausgaben des DSP über den Diagnosekanal zu setzen.
3	Ausgang logisch 0	Es soll möglich sein, am Ausgang dauerhaft eine logische Null zu senden. Über die Antenne würde dann nur das 3,951 MHz Signal gesendet werden.
4	Ausgang logisch 1	Es soll möglich sein, am Ausgang dauerhaft eine logische Eins zu senden. Über die Antenne würde dann nur das 4,516 MHz Signal gesendet werden.
5	Ausgang alternierend (0xAA…0xAA)	Es soll eine alternierende Bitfolge am Ausgang ausgegeben werden. Dabei entspricht die Bitfolge der maximalen Frequenz des Ausgangs: …1010101010101010…
6	Ausgang alternierend (0xF0…0xF0)	Es soll eine alternierende Bitfolge am Ausgang ausgegeben werden. Dabei entspricht die Bitfolge einem viertel der maximalen Frequenz des Ausgangs: …1111000011110000…
7	Ausgang alternierend (0xFF…0x00)	Es soll eine alternierende Bitfolge am Ausgang ausgegeben werden. Dabei entspricht die Bitfolge einem achtel der maximalen Frequenz des Ausgangs: …1111111100000000…
8	Beliebige Daten zyklisch senden	Es sollen beliebige Daten zyklisch gesendet werden können.
9	Telegramm zyklisch senden	Es soll möglich sein, ein Telegramm zyklisch zu senden. Dabei muss noch festgelegt werden, ob das zu sendende Telegramm einem Telegrammspeicherplatz des DSP entnommen wird, oder ob zu diesem Zweck ein festes Telegramm im Speicher abgelegt wird, das nicht geändert werden kann.

Nr.	Schlagwort	Beschreibung
10	Encoder Performance Test	Es soll möglich sein, einen Performance Test für den Encoder zu starten. Dafür muss die Anzahl der testweise zu kodierenden Telegramme als Parameter übergeben werden. Unter Umständen muss eine Funktion geschaffen werden, einen begonnenen Test auch abbrechen zu können (problematisch, da dazu eigener Task für RS 232 Protokoll notwendig).
11	Firmware Version	Es soll die aktuelle Softwareversion der Anwendung ausgelesen werden können.
12	TlgHeap lesen	Es soll möglich sein, den Status des Telegrammspeichers auszulesen. Dabei soll der Status aller Telegrammspeicherplätze zurückgegeben werden (Telegramm vorhanden / kodiert / unkodiert…)
13	SqcHeap lesen	Es soll möglich sein, den Status des Sequenzspeichers auszulesen. Dabei soll der Status aller Sequenzspeicherplätze zurückgeliefert werden (Sequenz vorhanden / Sequenzlänge / Sequenzeinhalt).
14	TX Fehlermodell lesen	Es soll möglich sein, das aktuell aktivierte TX Fehlermodell zu lesen.
15	TX Fehlermodell setzen	Es soll das TX Fehlermodell über den Diagnosekanal neu gesetzt werden können.
16	Telegram definieren	Es sollen Telegramme über den Diagnosekanal im DSP editiert werden können.
17	Telegram kodieren	Es sollen Telegramme über den Diagnosekanal kodiert werden können.
18	Sequenz definieren	Es soll möglich sein, eine Sequenz im DSP zu definieren
19	Sequenz starten	Es sollmöglich sein, eine Sequenz im DSP zu starten (aus dem Sequence Heap).

Für die Kommunikation über den Diagnosekanal soll ein Klartextprotokoll verwendet werden, da im Diagnosemodus keine zeitlichen Restrektionen für den DSP gesetzt sind. Ein Klartextprotokoll hat den großen Vorteil, dass es für den Benutzer einfacher zu verstehen und intuitiv bedienbar ist.

Dabei soll der Aufruf eines Vorganges über ein Kommando im Stile von *„LeseTelegramm-Speicher“*, oder *„SchreibeTelegrammDaten“* erfolgen. Alle weiteren Eingaben werden dann nach Aufruf des Kommandos vom DSP abgefragt.

Die einzelnen Kommandos sind nach dem Start selbst dafür verantwortlich, dass die Eingaben des Benutzers sinnvoll und vollständig sind. Sollten ungültige oder unsinnige Eingaben erfolgen, so sollen die Kommandos abgebrochen werden.

Zusätzlich muss es jederzeit möglich sein, alle Kommandos, die auf Eingaben des Anwenders warten, durch verlassen des Diagnosemodus zu verlassen.

6.2 Umsetzung

Die Implementation des Diagnosekanals erfordert einige Änderungen der bestehenden DSP - Software. Durch die Implementation eines zweiten RS 232 Protokolls, das quasi „gleichzeitig“ zum RailSiTe - Protokoll laufen soll, kann die bisherige Behandlung des RailSiTe - Protokolls (siehe Kapitel 3.2.2) so nicht erhalten bleiben.

Da es nicht möglich ist, beide Protokolle wirklich gleichzeitig ablaufen zu lassen, muss eine Umschaltung zwischen den beiden Protokollen möglich sein. Diese Umschaltung soll dadurch gegeben sein, dass der Anwender jederzeit den Diagnosemodus aktivieren kann. Im Diagnosemodus werden alle vom RailSiTe an den DSP gesendeten RS 232 Pakete vom DSP empfangen, aber anschließend mit einem **ACK(TX_INGORE)** verworfen. Der DSP reagiert in dieser Zeit nur auf Kommandos im Diagnosekanal.

Erst nach Beenden des Diagnosemodus durch schließen des Diagnosekanals wird das RailSiTe - Protokoll wieder abgearbeitet.

Die Umschaltung zwischen Diagnosemodus und RailSiTe - Modus muss also jederzeit möglich sein (auch wenn der DSP gerade auf Eingaben des Anwenders wartet). Aus diesem Grund wird die vorhandene Interrupt Service Routine (ISR) für den UART (Universal Asynchronous Repeater Transmitter, hier Peripheriebaustein auf der Daughtercard zur Kommunikation über RS 232) folgendermaßen geändert:

- ⇨ Ständige Überwachung des Diagnosekanals auf ASCII Zeichen **STRG-D**:

  Tritt das Zeichen **STRG-D** im Diagnosekanal auf, so wird der Status des Diagnosekanals getoggelt. War der Diagnosemodus zum Zeitpunkt des Auftretens aktiv, so wird er beendet. War das RailSiTe - Protokoll aktiv, so wird der Diagnosemodus aktiviert.

- ⇨ Verwerfen aller Zeichen im Diagnosekanal:

  Ist der Diagnosemodus inaktiv, so werden alle Zeichen, die über den Diagnosekanal empfangen werden, verworfen. Sie werden nicht im Empfangspuffer des Diagnosekanals gespeichert. Das Zeichen **STRG-D** wird ebenfalls bei Empfang verworfen.

- ⇨ Ignorieren aller RailSiTe - Pakete:

  Wenn der Diagnosemodus aktiv ist, werden alle RailSiTe - Pakete empfangen, aber nach Empfang eines kompletten Pakets verworfen. Dazu wird der komplette Header ausgewertet und gespeichert. Anschließend wird auf alle restlichen Daten des Pakets gewartet, bis die Anzahl der empfangenen Bytes der Längenangabe im Telegrammheader entspricht. Als Antwort wird mit **ACK(TX_INGORE, RESULT_OK)** auf die Pakete geantwortet.

  So lange der Diagnosemodus aktiv ist, werden keine Daten, die über die RailSiTe - Schnittstelle empfangen werden, im Empfangspuffer der RailSiTe - Schnittstelle gespeichert. Dadurch kann das RailSiTe - Protokoll nach Schließen des Diagnosekanals nahtlos an den vorherigen Betrieb anknüpfen.

In dieser Struktur ist das Diagnoseprotokoll das bevorzugte Protokoll. Seine Abarbeitung hat Vorrang vor dem RailSiTe - Protokoll. Die Information über den aktuellen Status des Diagnosekanals wird in der Struktur **protocol_status_s** gespeichert. Sie hat folgenden Aufbau.

```
typedef struct
{
        uint8 diag_active_u8;          // boolean, if diagnostic Chanel is active
} t_protocolStatus_s;
```

Über den Inhalt dieser Struktur kann der DSP bestimmen, welches der Protokolle zurzeit aktiviert ist. Die Auswertung der Daten, die über die beiden Schnittstellen an den DSP gesendet werden, findet in den Protokollhandlern statt.

Dabei existiert für jedes der beiden Protokolle (RailSiTe / Diagnose) ein eigener Protokollhandler. Der Aufruf des passenden Handlers zum jeweils aktuell laufenden Protokoll findet in der Funktion **mainLoop (…)** statt.

Dieser Vorgang stellt eine Neuerung in der DSP - Software zum alten Stand ohne Diagnosemodus dar. Dort wurde in der **mainLoop (…)** ausschließlich das RailSiTe Protokoll ausgewertet. Die Struktur der Protkollauswertung ist in Abbildung 35 gezeigt.

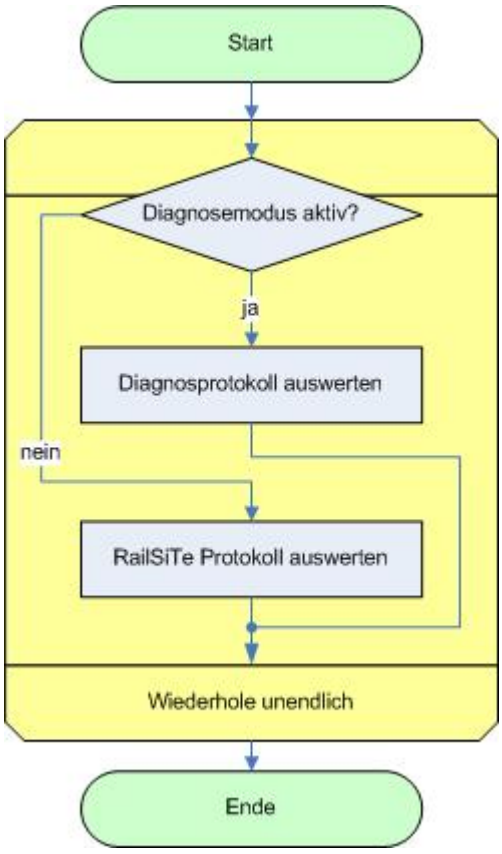

Abbildung 35: Flussdiagramm mainLoop (…)

Da beide Protokolle in der Restrechenzeit des DSP laufen, also nicht in einen extra Task implementiert sind, ist der DSP während des Wartens auf Daten über die RS 232 Schnittstellen zu 100 % ausgelastet. Um es der DSP - Software trotzdem zu ermöglichen, bei einer Umschaltung zwischen den beiden Protokollen in der UART ISR schnell das passende Protokoll zu starten, müssen alle Funktionen zum Empfangen von Daten über die Schnittstellen so angepasst werden, dass sie abgebrochen werden, falls zwischen den beiden Protokollen umgeschaltet wird.

Für die Auswertung des Diagnoseprotokolls müssen zusätzlich neue Funktionen implementiert werden. All diese Änderungen sollen in den nächsten Kapiteln näher erläutert werden.

Die Unterfunktionen der einzelnen Kommandos werden nicht einzeln aufgeführt. Eine Übersicht über alle Kommandos und deren Funktionalität wird aber in Kapitel 6.3 aufgeführt.

6.2.1 Funktion wait4UARTData (…)

Diese Funktion wartet auf den kompletten Empfang eines Datenblocks von der RailSiTe - Schnittstelle des DSP. Dabei werden als Parameter ein Zeiger auf den Puffer übergeben, in dem die Daten gespeichert werden sollen und die Anzahl der zu empfangenden Bytes.

Den Rückgabewert bildet entweder die Anzahl der tatsächlich empfangenen Bytes, oder eine Fehlerkonstante.

Tabelle 38: Parameter der Funktion wait4UARTBlock (…)

Name	Typ	Beschreibung
buffer_pu8	uint8*	Zeiger auf den Datenpuffer für empfangene Daten
length_u32	uint32	Anzahl der Bytes, auf die gewartet werden soll.

Tabelle 39: Rückgabewert der Funktion wait4UARTBlock (…)

Name	Typ	Beschreibung
Return	uint32	Entweder Anzahl der empfangenen Bytes, oder Fehlercode: RS232_PAKET_TIMEOUT: Timeout bei warten auf Daten RS232_PAKET_DIAG_MODE: Umschaltung in Diagnosemodus während warten auf Daten.

Die Funktion **wait4UARTData (…)** muss in der Lage sein, das Warten auf Daten abzubrechen, wenn der Diagnosekanal aktiviert wurde. In diesem Fall wird die Funktion mit dem Rückgabewert **RS232_PAKET_DIAG_MODE** verlassen.

Weiterhin muss die Funktion so erweitert werden, dass nicht unendlich lange auf Daten gewartet wird, da das RailSiTe - Protokoll im DSP und im RailSiTe sonst asynchron laufen, wenn neue Daten vom RailSiTe an den DSP gesendet werden, während der noch auf alte Daten wartet, die unter Umständen bei der Übertragung verloren gegangen sind.

Hierzu wird ein Timeout implementiert, der es erlaubt, den Empfang von Daten nach dem Ablauf einer definierten Zeitperiode abzubrechen und das begonnene Paket auf der RailSiTe - Schnittstelle mit einem **ACK (TX_IGNORE, RESULT_OK)** zu quittieren. Dadurch weiß das RailSiTe - Labor, dass das Paket fehlerhaft empfangen wurde und er DSP ist mit Start des Empfangs des nächsten Pakets vom RailSiTe wieder mit diesem synchron.

Um für den Timeout eine definierte Zeit einstellen zu können, wird der Timer 0 des DSP so konfiguriert, das er alle 1 ms einen Interrupt auslöst. Bei Auftritt dieses Interrupt wird die Funktion **timer0_ISR (…)** aufgerufen. Dort wird der Timeout Zähler inkrementiert.

In der Funktion **RS232_PAKET_TIMEOUT** wird der Timeout Zähler zu Beginn der Wartezeit auf die Daten auf den Wert 0 gesetzt. Anschließend wird ständig auf die Konstante **RS232_RAILSITE_TIMEOUT** verglichen. Wenn der Wert des Timeoutzählers diesen Wert überschreitet, so wird die Funktion **wait4UARTData (…)** mit der Fehlerkonstante **RS232_PAKET_TIMEOUT** verlassen. Durch dieses Vorgehen kann der Timeoutwert für die Funktion millisekunden genau eingestellt werden.

Außerdem erlaubt der Timer 0 die Implementation einer millisekunden genauen Systemuhr.

Eine Übersicht über die Funktion **wait4UARTData (…)** gibt Abbildung 36.

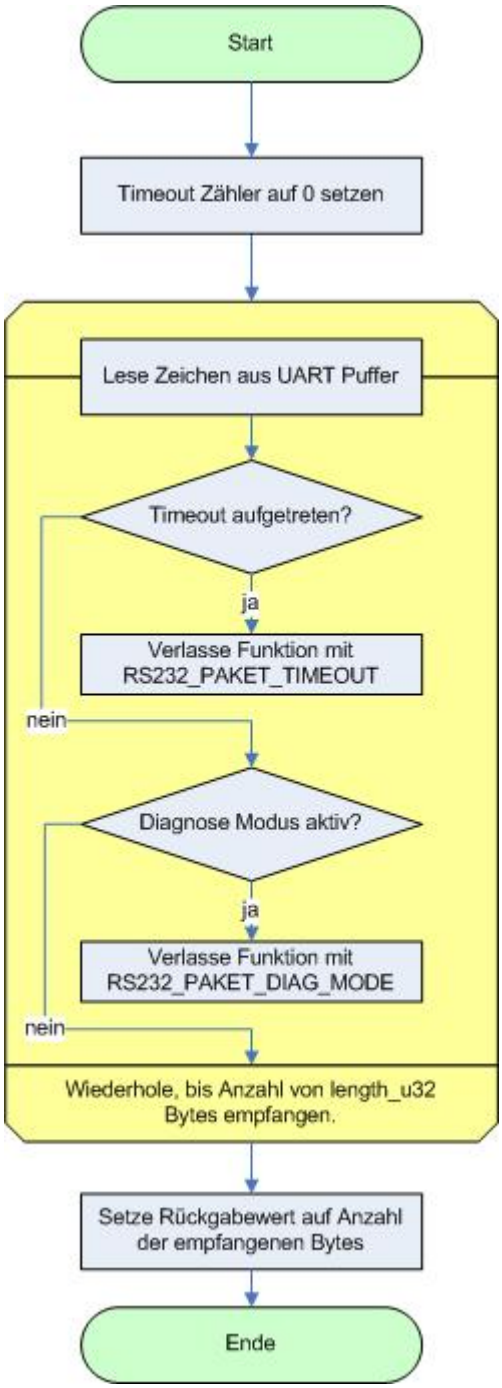

Abbildung 36: Flussdiagramm wait4UARTData (…)

6.2.2 Funktion wait4UARTString (…)

Die Funktion **wait4UARTString (…)** bildet das Äquivalent zu **wait4UARTData (…)** im Diagnosekanal. Sie erhält als Übergabeparameter allerdings nur einen Verweis auf den Empfangspuffer, da alle Strings, die über den Diagnosekanal gesendet werden, mit einem Endezeichen (RETURN) beendet werden.

Als Rückgabewert gibt die Funktion entweder die Länge des empfangenen Strings, oder einen Fehlerkode zurück.

Tabelle 40: Parameter der Funktion wait4UARTString (…)

Name	Typ	Beschreibung
string_pu8	uint8*	Zeiger auf den Datenpuffer für empfangene Daten

Tabelle 41: Rückgabewert der Funktion wait4UARTString (…)

Name	Typ	Beschreibung
Return	uint32	Entweder Anzahl der empfangenen Bytes, oder Fehlercode: DIAG_MODE_DEACTIVATED: Diagnosemodus wurde verlassen DIAG_REC_BUFFER_OFL: empfangene Zeichenkette ist länger als der Empfangspuffer.

Die Funktion **wait4UARTString (…)** wartet nach dem Aufruf auf den Empfang eines kompletten Kommandos über den Diagnosekanal. Dabei werden so lange Zeichen aus dem Empfangspuffer der Diagnoseschnittstelle ausgelesen und im String gespeichert, bis das Endezeichen (RETURN) empfangen wurde.

Eine weitere Bedingung für den Abbruch des Empfangs von Zeichen ist ein Überlauf im String. Dazu wird nach jedem empfangenen Zeichen überprüft, ob bereits mehr als **DIAG_CMD_MAX_LEN** Zeichen empfangen wurden. Diese Konstante gibt die Länge in Byte des Puffers an, in dem alle über die Diagnoseschnittstelle empfangenen Strings gespeichert werden. Sollten mehr als diese Zeichen empfangen werden, so wird die Funktion **wait4UARTString (…)** mit dem Fehler **DIAG_REC_BUFFER_OFL** verlassen. Dieses Vorgehen verhindert einen Überlauf des Empfangspuffers.

Außerdem wird in jedem Durchlauf überprüft, ob der Diagnosekanal noch aktiv ist. Sollte der Diagnosemodus verlassen werden und das RailSiTe - Protokoll wieder aktiv sein, so wird der Empfang der Daten über den Diagnosekanal abgebrochen. Alle bereits empfangenen Daten werden dabei verworfen.

Eine Übersicht über die Funktion **wait4UARTString (…)** bietet Abbildung 37.

Die Funktion **wait4UARTString_NoEndchar (…)** hat die gleiche Funktionalität, wartet allerdings nicht auf ein Endezeichen, sondern auf den Empfang einer bestimmten Anzahl von Bytes. Sie entspricht weitestgehend der Funktion **wait4UARTData (…)** in Kapitel 6.2.1 und wird deshalb hier nicht näher beschrieben.

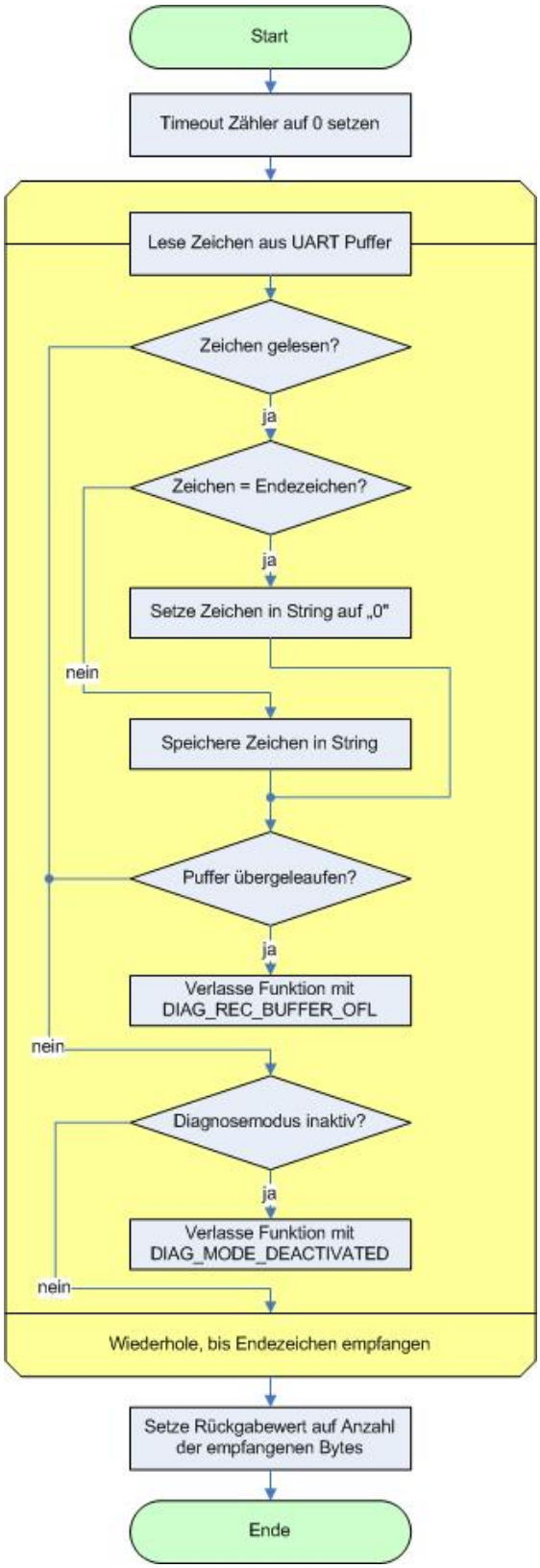

Abbildung 37: Flussdiagramm wait4UARTString (…)

6.2.3 Funktion handleDiagProtocol (…)

Die Funktion **handleDiagProtocol (…)** enthält die eigentliche Abwicklung des Diagnoseprotokolls. Sie wird zyklisch aus der Funktion **mainLoop (…)** aufgerufen, wenn der Diagnosemodus aktiviert ist. Sie besteht aus einer großen Liste, in der in die einzelnen Unterfunktionen zur Ausführung des Diagnoseprotokolls verzweigt wird. Sie besitzt weder Parameter, noch Rückgabewerte.

Tabelle 42: Parameter der Funktion handleDiagProtocol (…)

Name	Typ	Beschreibung
-	void	-

Tabelle 43: Rückgabewert der Funktion handleDiagProtocol (…)

Name	Typ	Beschreibung
-	void	-

Zu Beginn der Funktion **handleDiagProtocol (…)** wird auf den Empfang eines Diagnosekommandos gewartet. Wurde ein komplettes Kommando, inklusive Endezeichen empfangen, so wird in einer Liste überprüft, welches der Kommandos angesprochen wurde.

Dazu werden zuerst alle Zeichen des empfangenen Strings in kleine Zeichen (eng. Lower case) umgewandelt, um unempfindlich für Groß- und Kleinschreibung zu sein. Anschließend wird überprüft, ob beim Warten auf das Kommando ein Pufferüberlauf aufgetreten ist, oder der Diagnosemodus beendet wurde.

Im Falle eines Pufferüberlaufes wird eine Fehlermeldung an den Benutzer gesendet und das Kommando verworfen. Wurde während des Wartens auf ein Diagnosekommando der Diagnosemodus verlassen, so wird die Funktion **handleDiagProtocol (…)** ohne Fehlermeldung verlassen.

In allen anderen Fällen wurde ein Diagnosekommando korrekt bis zum Endezeichen empfangen und der empfangene String auf das angesprochene Kommando überprüft. Zu diesem Zweck sind alle gültigen Kommandos des Diagnosekanals als Konstanten im DSP hinterlegt. Enthält das empfangene Kommando einen unbekannten String, so wird eine Fehlermeldung an den Benutzer gesendet und das Kommando verworfen.

Bei Empfang eines gültigen Kommandos wird die zugehörige Unterfunktion aufgerufen, die den Ablauf des Kommandos übernimmt. Alle weiteren Ein- und Ausgaben über den Diagnosekanal erfolgen dann ausschließlich in den Unterfunktionen. Aus diesem Grund müssen die Unterfunktionen selbst dafür sorgen, dass sie im Falle des Wartens auf Eingaben des Benutzers in der Lage sind, auf eine Deaktivierung des Diagnosemodus zu reagieren.

Eine Übersicht über die Funktion **handleDiagProtocol (…)** zeigt Abbildung 38.

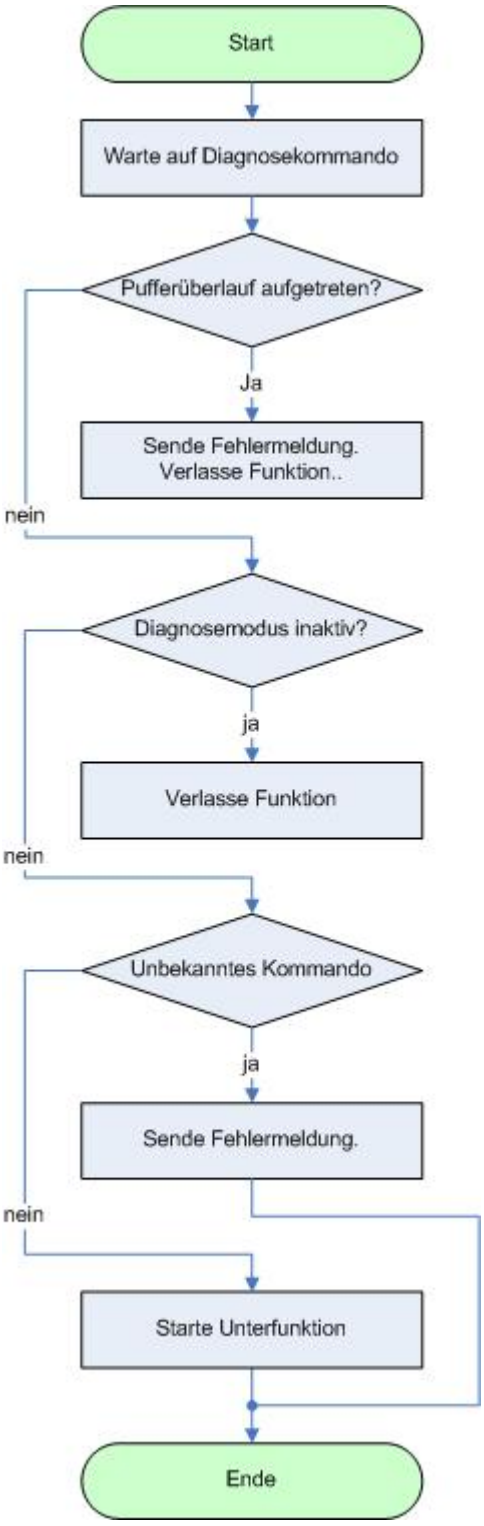

Abbildung 38: Flussdiagramm handleDiagProtocol (…)

6.3 Objektverzeichnis

Dieses Kapitel soll einen Überblick über alle zu implementierenden Kommandos des Diagnosekanals und deren anvisierte Funktionalität geben. Dabei werden die einzelnen Unterfunktionen nicht genauer beschrieben, sondern nur der Kommandoname und das Verhalten des DSP aufgeführt, dass dieser bei Empfang des Kommandos über den Diagnosekanal zeigen soll.

Tabelle 44: Kommandoübersicht Diagnosekanal

Name (String)	Beschreibung
getHelp	Bei Aufruf dieses Kommandos wird eine Tabelle mit allen verfügbaren Kommandos und deren Funktion an den Benutzer gesendet. Alle Strings hierzu werden als Konstanten in der DSP - Software angelegt. Dadurch können sie später einfach geändert werden (z.B. andere Sprachen).
doReset	Bei Aufruf dieses Kommandos wird die DSP - Software zurückgesetzt. Dazu wird die Funktion **initComponents (…)** aufgerufen, die alle Komponenten der DSP - Software neu initialisiert. Nach dem Aufruf dieses Kommandos befindet sich der DSP wieder im RailSiTe - Modus. Das Diagnoseprotokoll ist inaktiv.
setLogLevel	Über dieses Kommando kann der Loglevel zur Ausgabe von Debugmeldungen im RailSiTe - Modus gesetzt werden. Es stehen dazu folgende Werte zur Verfügung: 1: alle Meldungen anzeigen 2: nur Infos anzeigen 3: nur Warnungen anzeigen 4: nur Fehlermeldungen anzeigen
getTlgHeap	Mit diesem Kommando kann der Inhalt des kompletten Telegramm Speichers im DSP abgefragt werden. Die Ausgabe erfolgt in einer Baumstruktur, wobei die eigentlichen Telegrammdaten nicht mit dargestellt werden.
getTlgSlot	Bei Aufruf dieses Kommandos kann der Inhalt eines Telegrammspeicherplatzes im Telegramm Heap des DSPs gelesen werden. Dabei werden der Inhalt der unkodierten und kodierten Telegrammdaten mit übertragen.
setTlgSlot	Über dieses Kommando kann ein Telegrammspeicherplatz im DSP vom Benutzer beschrieben werden.
getSeqHeap	Über dieses Kommando kann der komplette Inhalt des Sequenz Speichers des DSP ausgelesen werden. Die Ausgabe erfolgt in einer Baumstruktur je nach Sequenz Speicherplatz und den darin gespeicherten Sequenzteilen.
getSeqSlot	Mit diesem Kommando kann ein einzelner Sequenzspeicherplatz vom Benutzer ausgelesen werden.
setSeqSlot	Bei Aufruf dieses Kommandos ist es dem Benutzer möglich, einen Sequenzspeicherplatz manuell zu beschreiben und so eine eigene Sequenz zu definieren.
startSeq	Mit diesem Kommando kann eine Sequenz des Sequenzspeichers des DSP gestartet werden.
getTXmod	Über dieses Kommando kann das aktuell aktivierte TX - Fehlermodell ausgelesen werden.
setTXmod	Mit diesem Kommando kann der Benutzer das TX - Fehlermodell neu setzen. Zur Verfügung stehen dabei alle Fehlermodelle, die in der Datei **txerror_functions.c** definiert sind.

Name (String)	Beschreibung
sendCyclData	Über dieses Kommando ist es dem Benutzer möglich, zyklisch bestimmte Daten über die HF - Schnittstelle (Balise) zu senden. Dabei stehen folgende Möglichkeiten zur Verfügung: 0: Sende Lowpegel (0) 1: Sende Highpegel (1) 2: Sende alternierende Bitfolge (maximale Ausgangsfrequenz 282,24 kHz) 3: Sende alternierende Bitfolge (halbe Ausgangsfrequenz 141,12 kHz) 4: Sende alt. Bitfolge (viertel der max. Ausgangsfrequenz 70,56 kHz) 5: Sende beliebige Daten (32Bit Wert vom Benutzer eingegeben) 6: Sende Telegramm aus Telegrammspeicher (Auswahl vom Benutzer) Dieses Kommando fast damit die Punkte 3 bis 9 der Tabelle 37 zusammen.
doEncTlg	Bei Aufruf dieses Kommandos hat der Benutzer die Möglichkeit, ein Telegramm aus dem Telegrammspeicher zu kodieren. Er kann dabei Initialwerte für SB und ESB eingeben, oder SB und ESB zufällig generieren lassen. Am Ende der Kodierung wird eine Übersicht über den Kodierungsvorgang ausgegeben.
doEncPerfTst	Über dieses Kommando kann der Benutzer einen Encoder Performance Test starten. Dazu muss er die Anzahl der zu kodierenden Telegramme, das Telegrammformat (long / short) und die Startwerte für SB und ESB an den DSP übertragen. Anschließend wird der Test gestartet und mit der Ausgabe einer Übersicht beendet. **Wichtig:** Der Test kann nach dem Start nur durch einen Hardwarereset des DSP beendet werden. Es sollte also bei Auswahl der Anzahl der zu kodierenden Telegramme darauf geachtet werden, dass der Test unter Umständen sehr lange dauern kann. **Tip:** Die Ergebnisse lassen sich einfach in Excel auswerten, wenn die RS 232 Übertragung während des Tests in einem Logfile mitgeschrieben wird.
getSysTime	Dieses Kommando liefert nach dem Aufruf den aktuellen Stand der Systemuhr zurück. Diese wird bei jedem Reset zurückgesetzt.

Die Namen aller Kommandos werden als Konstanten in der DSP - Software angelegt, so dass sie bei Bedarf leicht an zentraler Stelle geändert werden können.

Beim Anlegen neuer Kommandos müssen die Funktionen **handleDiagProtocol (…)** und **getHelp (…)** angepasst werden.

7 Aufbau eines neuen FSK Modulators

7.1 Anforderungen

Der schon in Kapitel 3.2.1 beschriebene Modulator zur Umsetzung der digitalen Datensignale in analoge Signale hat in der jetzigen Form einen großen Nachteil. Bei der Modulation entstehen durch die harte Umtastung zwischen zwei extern generierten Frequenzen große Phasensprünge. Diese können dazu führen, dass das analoge Signal im Empfänger nicht fehlerfrei demoduliert und so nicht empfangen werden kann.

Aus diesem Grund wurde in Rahmen der Diplomarbeit ein neuer Modulator erdacht, der eine phasenlineare Frequenzmodulation durchführt. Abbildung 39 zeigt den Unterschied zwischen den Ausgangssignalen der beiden Modulatoren.

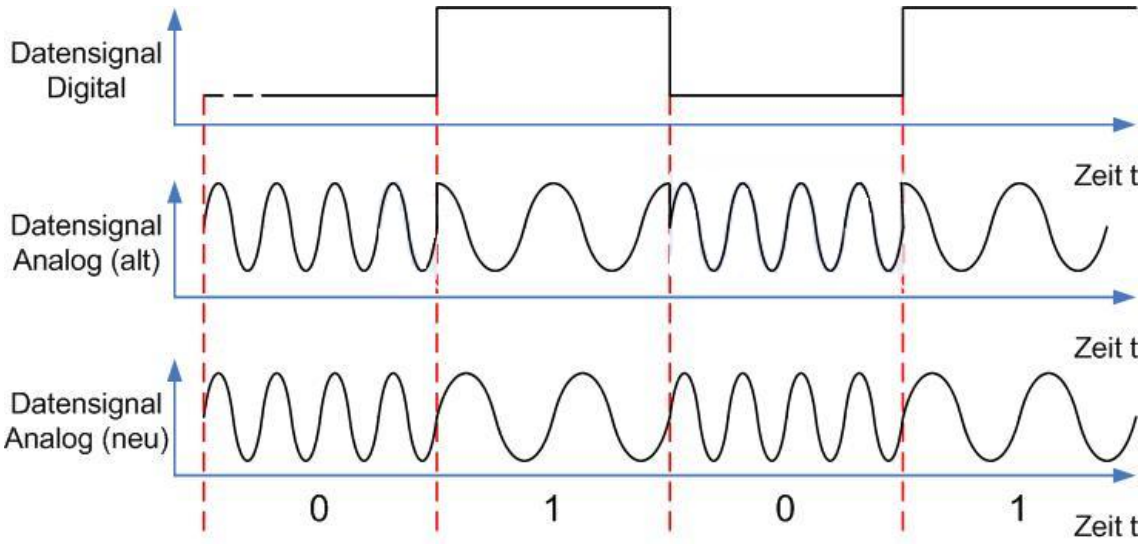

Abbildung 39: Ausgangssignal FSK-Modulator

Die Anforderungen an die neue Schaltung ergeben sich also aus der nötigen stetigen Phase des Ausgangssignals, der maximalen Frequenz des Modulationsignals und den beiden für die Modulation benötigten Frequenzen.

Die maximale Frequenz des Modulationssignals ergibt sich aus der Datenrate R, mit der die Daten über die HF - Schnittstelle übertragen werden. Diese Datenrate liegt bei 564,48 kBit/s. Die maximale Frequenz des digitalen Modulationsignals, dass im Modulator umgesetzt werden muss beträgt nun genau die Hälfte der Datenrate R, da als höchste Frequenz eine alternierende High / Low Folge am Modulatoreingang ausgegeben werden kann.

$$f_{ModMax} = \frac{R}{2} = \frac{564{,}48 \cdot 10^3 \cdot Bit}{2 \cdot s} = 282{,}24kHz \qquad \textbf{(11)}$$

Formel 11: maximale Frequenz des Modulationssignals

Die bei der Frequenzumtastung (Frequency Shift Keying FSK) benutzten Frequenzen betragen 3,951 MHz für eine logische Null und 4,516 MHz für eine logische Eins des Modulationssignals.

Die Anforderungen an den neuen FSK Modulator lassen sich also wie folgt formulieren:

- ⇨ stetiger Phasenverlauf bei Frequenzumtastung
- ⇨ minimale Modulationsfrequenz von 282,24 kHz
- ⇨ Modulation mit Frequenzen im Bereich von 3 MHz bis 5 MHz
- ⇨ Einfache Ansteuerung der Schaltung (möglichst nur über die Datensignale des DSP DATA / GATE)
- ⇨ Platinenlayout geeignet zur Montage auf das bestehende DSP - Board
- ⇨ möglichst große Amplitude und geringer Klirrfaktor des Ausgangssignals mit guter Unterdrückung des Ausgangssignals im abgeschalteten Zustand (GATE = Low)

Die Forderung nach Einhaltung des Referenzfeldes, das durch die ETCS Spezifikation für Eurobalisen vorgeschrieben wird, wird beim Aufbau dieses Modulators wissentlich ignoriert. Um das Referenzfeld nachzubilden, müsste das Signal in Rampen hochgefahren werden, die von der simulierten Geschwindigkeit und Position des Zuges abhängig sind. Abbildung 40 zeigt das Referenzfeld für Eurobalisen.

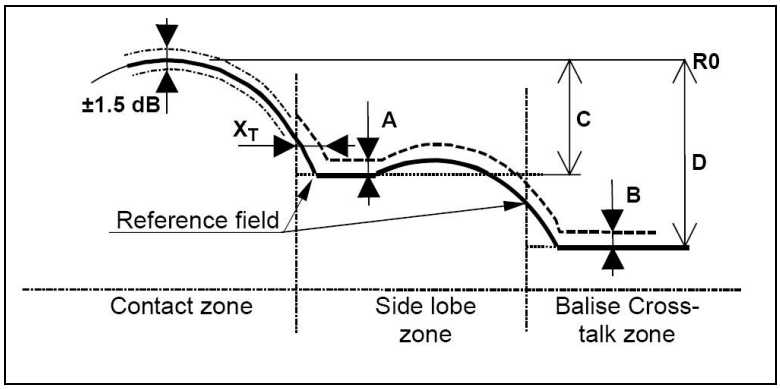

Abbildung 40: Eurobalise Referenzfeld (Quelle: Subset-036, Seite 68 Abb. 14)

Im Gegensatz zu diesem Referenzfeld erfolgt die Übertragung des DSP - Systems über den Weg praktisch als Rechteckfunktion. Die Sendeleistung liegt sofort bei Einfahrt des Zuges in das simulierte Zielfenster mit vollem Pegel an.

Aus diesem Grund ist im Fahrzeugrechner keine genaue Bestimmung des Maximums des Sendefeldes der Eurobalise möglich, über das im Normalfall der Balisenmittelpunkt bestimmt wird. Vielmehr wird der Balisenmittelpunkt schon am Anfang des Sendefensters vom Fahrzeugrechner festgestellt.

Daraus resultiert ein Wegfehler bei der simulierten Balisenübertragung, der allerdings konstant die Hälfte des Sendefensters beträgt. Bei der derzeit in der RailSiTe - Software benutzten Fensterbreite von 1 m beträgt der Fehler demnach 0,5 m.

Sollte dieser Fehler zu Problemen in der Balisenanbindung führen, muss er in der BTM - Software korrigiert werden, da nur das BTM sowohl über das Wissen der Geschwindigkeit des Zuges, als auch der Position des Zuges und der Balisen verfügt. Eine Korrektur in der DSP - Software ist nicht möglich, da dort die Information über die aktuelle Geschwindigkeit des Zuges fehlt, die zur Berechnung eines zeitlichen Korrekturwertes notwendig ist.

7.2 Umsetzung

7.3 modularer Aufbau der Schaltung

Auf Grund der in Kapitel 7.1 an den neuen Modulator gestellten Anforderungen wird die Schaltung aus drei wesentlichen Komponenten aufgebaut. Abbildung 41 soll den Aufbau verdeutlichen.

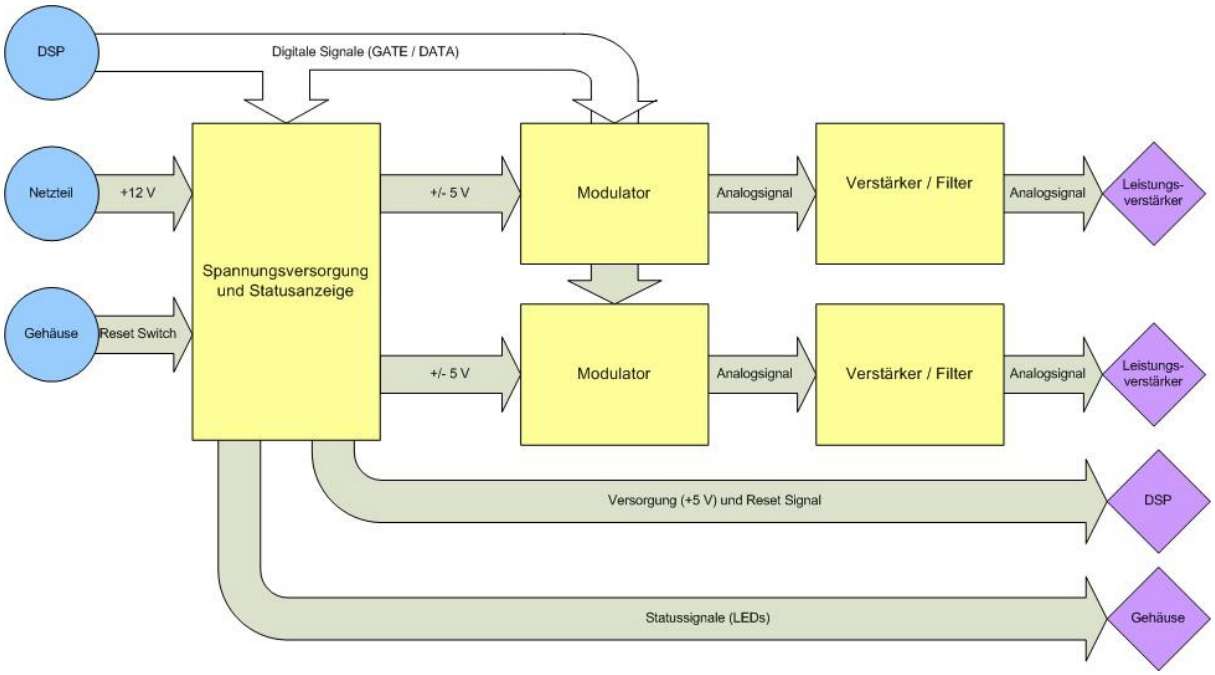

Abbildung 41: modularer Aufbau der Modulator Platine

Alle drei Komponenten sollen auf einer einzelnen Platine implementiert werden, die später einfach auf das bereits bestehende System aufgesetzt wird.

Grundsätzlich sollen alle Schnittstellen zwischen den Komponenten so modular wie möglich aufgebaut werden. Das gilt insbesondere für die Schnittstellen zwischen der Platine und dem DSP - Board, bzw. dem Gehäuse. Dieses Vorgehen bietet den Vorteil, dass einzelne Komponenten einfach ausgetauscht und erweitert werden können und z.B. keine Hardwareänderungen am DSP - Board durchgeführt werden müssen.

Bei Änderungen an den Komponenten, die auf der Platine integriert werden, muss zwar das Layout für das einzelne Modul angepasst werden, alle anderen Komponenten können aber ohne größere Änderungen in das neue Layout übernommen werden.

Die einzelnen Module werden nachfolgend kurz beschrieben und in den nächsten Kapiteln genauer erläutert.

⇨ **Modulator:**

Das Modulatormodul soll die eigentliche Umsetzung des digitalen Datensignals, dass durch den DSP über den McBSP ausgegeben wird, in ein analoges Signal bewerkstelligen, das über eine Antenne an den Fahrzeugrechner gesendet werden kann. Da dieser Schaltungsteil für jeden der beiden RailSiTe Kanäle benötigt wird, wird er doppelt ausgelegt.

⇨ **Ausgangsfilter und Verstärker:**

Der Verstärker soll dafür sorgen, dass das Ausgangsignal vom Modulator entkoppelt wird und zusätzlich die Amplitude des Ausgangssignals ausreichend groß. Er wird als aktiver Hochpassfilter ausgelegt, um den Ausgang des Modulators stumm zu schalten, falls das GATE Low ist, also aktuell keine Daten gesendet werden.

⇨ **Spannungsversorgung und Statusanzeige:**

Da das komplette DSP - System später als autarkes Gerät in Form einer Blackbox aufgebaut werden soll, wird auf die Modulatorplatine eine Spannungsversorgung integriert. Sie soll aus einer Eingangsspannung sowohl die +5 V und -5 V Versorgungsspannung für den Modulator, als auch die +5 V Versorgungsspannung für das DSP - Board zur Verfügung stellen. Zusätzlich soll dieses Modul über LEDs über den Status des DSP - Systems informieren.

7.4 Aufbau des Modulators

Der Modulator sorgt für die Umsetzung der Digitalsignale des DSP in analoge Signale, die später über die Antenne an das reale Fahrzeuggerät übertragen werden können. Zur Erzeugung des analogen FSK - Signals soll eine integrierte Schaltung (Integrated Circuit IC) als Basisbaustein für die Modulation genutzt werden, in dem die gesamte Umsetzung des digitalen Modulationssignals in ein analoges Ausgangssignal gekapselt wird.

Die Wahl fiel dabei auf den Baustein MAX 038 der Firma MAXIM. Dieser IC beherrscht die Erzeugung von analogen Sinusschwingungen in einem Bereich von 0,1 Hz bis 20 MHz mit einer Amplitude von maximal 2V Spitze / Spitze.

Dabei kann die Frequenz des Ausgangssignals wahlweise durch eine dem Modulationssignal (DATA) entsprechende Spannung, oder einen Strom eingestellt werden. Die maximale Frequenz des Modulationssignals darf bis zu 2 MHz betragen. Im Ausgangssignal treten trotz Sprüngen im Modulationssignal keine Phasensprünge auf. Auf Grund der einfachen Ansteuerung und der hohen Modulationsfrequenz eignet sich dieser IC ideal für die Verwendung als FSK Modulator für die Balisenansteuerung. Eine Übersicht über den Baustein ist im Blockschaltbild in Abbildung 42 gegeben.

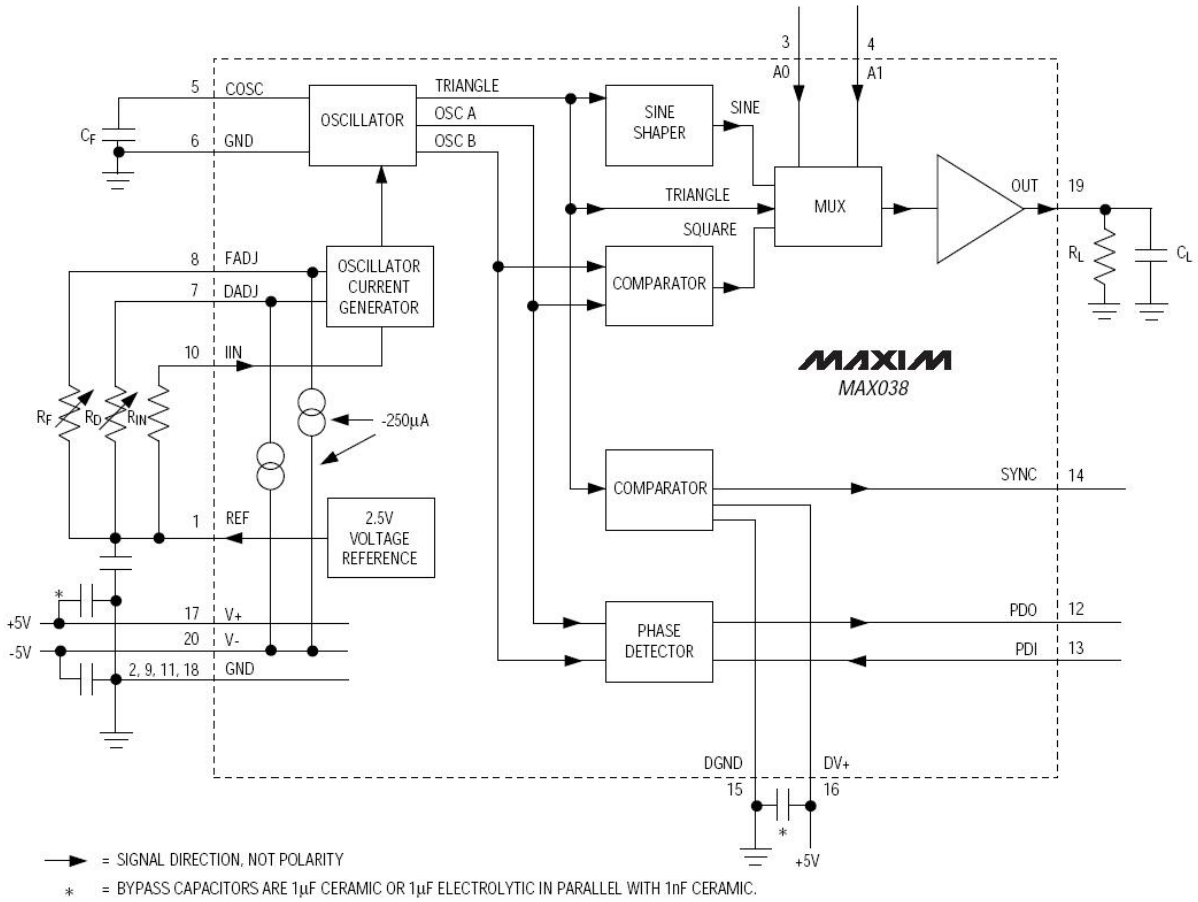

Abbildung 42: Blockschaltbild MAX038 (Quelle: [8], Seite 8, Abb. 1)

Als Eingangssignal wird in diesem Aufbau ein Strom in den MAX 038 eingespeist, der aus der Referenzspannungsquelle des ICs und mehreren Widerständen erzeugt wird. Der Eingangsstrom darf sich im Bereich von 2,5 µA bis 750 µA bewegen. Dieser Strom hat unterschiedliche Werte für ein Low- und einen Highpegel des Modulationssignals DATA.

Die Ansteuerung des MAX 038 erfolgt über einen Analogschalter IC C4070, der die einzelnen Kanäle von V_{REF} des MAX 038 über die für die Frequenzeinstellung nötigen Widerstände auf den Eingang I_{in} des MAX 038 schaltet. Als Kontrollsignale werden dazu jeweils das echte und invertierte Signal des Modulationssignals DATA verwendet.

Die Invertierung des Modulationssignals erfolgt über einen Logik IC C4016 (4fach XOR). Dabei wird sowohl das invertierte, als auch das nicht invertierte Signal des Modulationssignals einmal durch das Gatter geführt, um Laufzeitunterschiede zu vermeiden. Das Modulationssignal wird dazu jeweils mit LOW, oder HIGH Pegel verknüpft, um die resultierenden Signale zu erhalten. Dabei gilt:

$$DATA = DATA \text{ xor } 0$$

$$\overline{DATA} = DATA \text{ xor } 1$$

Der so erzeugte Strom zur Einstellung der Frequenz des MAX038 wird, bevor er in den IIN Eingang des MAX038 eingespeist wird, nochmals über den Analogschalter IC geleitet. Dort wird es mit dem GATE Signal verknüpft, so dass der Strom immer dann in den MAX038 gespeist wird, wenn Daten gesendet werden. Für den Fall, dass keine Daten gesendet werden, das GATE Signal also Low Pegel besitzt, wird der Strom über einen parallel liegenden Widerstand im Megaohm Bereich stark herabgesetzt, um in diesem Fall die Ausgangsfrequenz herabzusetzen. Das Restsignal kann im diesem Fall im nachfolgenden Filter beseitigt werden.

Im nachfolgenden Bild soll das grundsätzliche Verhalten des Modulators als Zustandsdiagramm dargestellt werden.

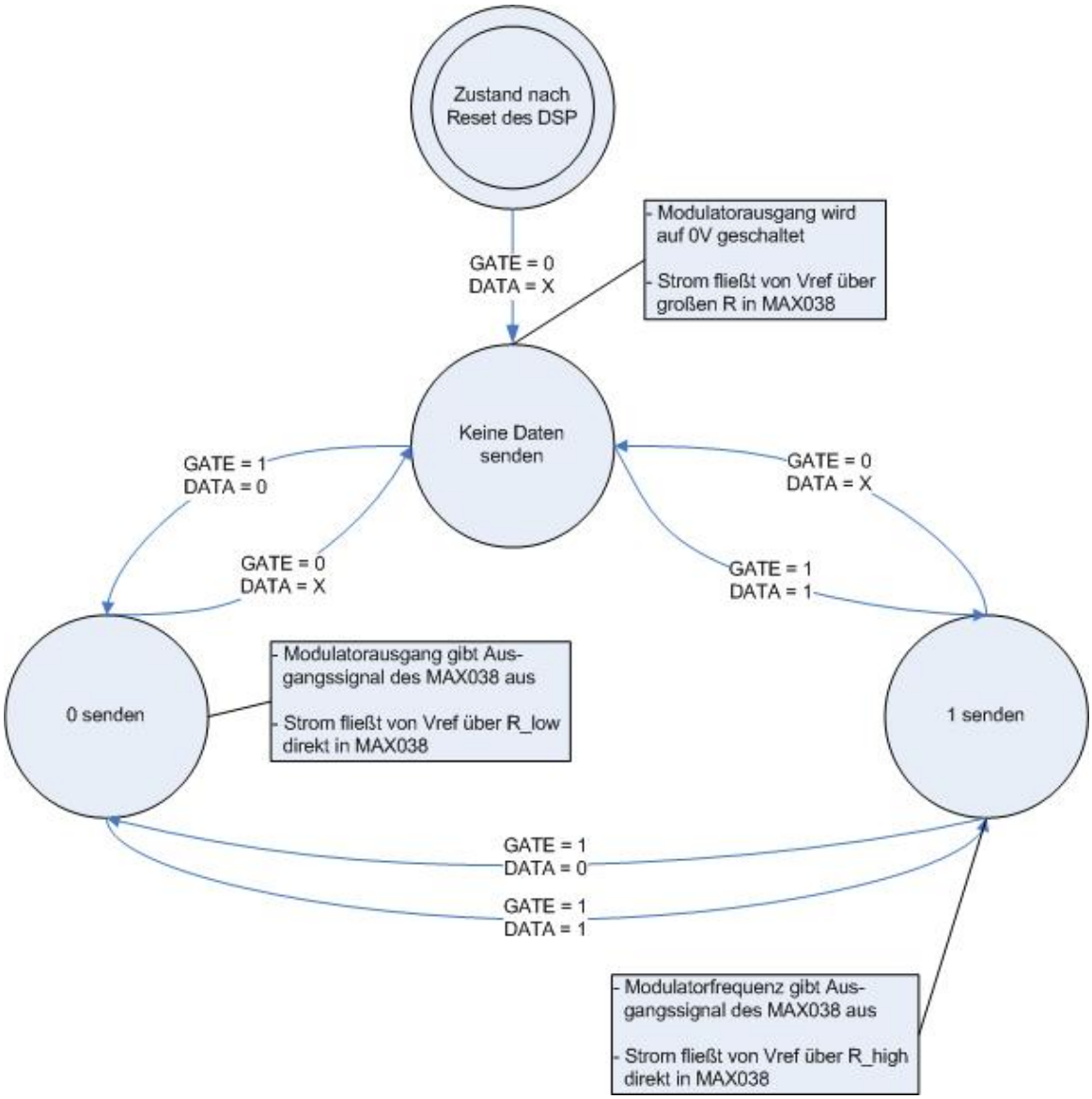

Abbildung 43: Zustandsdiagramm des FSK-Modulators (neu)

Um die Arbeitspunkte des MAX 038 festzulegen, müssen die Vorwiderstände für den Eingangsstrom und der Kondensator zur Festlegung des Frequenzbereiches berechnet werden. Für die Berechnung der Bauelemtwerte muss folgende Formel verwendet werden (siehe [8], Seite 11).

$$f = \frac{2 \cdot U_{REF}}{R_{IN} \cdot C_F}$$

mit $\frac{U_{REF}}{R_{IN}} = I_{IN}$ folgt

$$C_F = \frac{2 \cdot I_{IN}}{f_{Out}} \qquad \textbf{(12)}$$

Formel 12: Berechnung des Kondensators C_F für FSK Modulator

Der mögliche Frequenzraum des MAX 038 wird dabei durch die Kapazität C_F festgelegt. V_{REF} entspricht der Referenzspannung des MAX 038 von 2,5 V. R_{IN} ist dann der Widerstand, der den Strom von der Referenzspannungsquelle V_{REF} zum Eingang I_{IN} des MAX 038 festlegt. Nach einer Einheitenprüfung ist festzustellen, dass die Einheiten der resultierenden Kapazität nicht der Einheit Fahrad entsprechen. Auf Anfrage bei Maxim ist diese Formel trotzdem korrekt und gilt nur als näherungsweise Berechnungsvorschrift für den Wert von C_F. In den nachfolgenden Berechnungen werden deshalb die Einheiten bewusst nicht berücksichtigt.

Als Vorgabe für die Schaltung soll gelten:

⇨ Eingangstrom $I_{IN} = 255{,}0\mu A$ für die Mittenfrequenz von $f_{OutMid} = 4{,}200MHz$

Dann folg mit Formel 12 für den Wert von C_F:

$$C_F = \frac{2 \cdot I_{INMid}}{f_{OutMid}} = \frac{2 \cdot 255{,}0 \cdot 10^{-6}}{4{,}200 \cdot 10^{6}} = 121{,}429 \cdot 10^{-12} \qquad \textbf{(13)}$$

Formel 13: Berechnung der Kapazität C_F

Da der nächste erhältliche Kapazitätswert 120,0 pF entspricht, wird $C_F = 120{,}0\,pF$ festgelegt. Aus der Kapazität C_F und den beiden Ausgangsfrequenzen für High- und Low Pegel des Datensignals DATA lassen sich die notwendigen Widerstände R_{IN} berechnen.

$$R_{INHigh} = \frac{2 \cdot U_{REF}}{f_{OutHigh} \cdot C_F} = \frac{2 \cdot 2{,}5}{4{,}516 \cdot 10^{6} \cdot 120 \cdot 10^{-12}} = 9{,}2265k\Omega$$

$$R_{INLow} = \frac{2 \cdot U_{REF}}{f_{OutLow} \cdot C_F} = \frac{2 \cdot 2{,}5}{3{,}951 \cdot 10^{6} \cdot 120 \cdot 10^{-12}} = 10{,}5459k\Omega \qquad \textbf{(14)}$$

Formel 14: Berechnung der Widerstände R_{IN} für FSK Modulator

Die Widerstände werden auf einen gemeinsamen Vorwiderstand von 8,2 kΩ und jeweils ein Trimmpotentiometer von 2,0 kΩ zur Justierung der Ausgangsfrequenz aufgeteilt. Für den Zweig des Ausgangssignals bei Low Pegel des Datensignals muss ein zusätzlicher 1,2 kΩ Widerstand vorgesehen werden.

Um für den Fall von einem Low Pegel im GATE Signal den Ausgang des Modulators einfach stumm schalten zu können (einfache Auslegung des nachfolgenden Filters), soll der Widerstand für den Parallelzweig so berechnet werden, dass der minimale Strom von 2,5 µA in den Eingang IIN des MAX 038 fließt. Dadurch liegt am Ausgang des MAX 038 die minimal mögliche Frequenz an.

$$R_{INOff} = \frac{U_{REF}}{I_{INOff}} = \frac{2{,}5}{2{,}5 \cdot 10^{-6}} = 1 \cdot 10^{6} \qquad \textbf{(15)}$$

Formel 15: Berechnung des Widerstands R_{IN} für (GATE = LOW)

In diesem Fall erzeugt der MAX 038 eine Signal mit einer Amplitude von 2,0 V Spitze / Spitze und einer Frequenz von:

$$f_{OutOff} = \frac{2 \cdot V_{\mathrm{Re}f}}{R_{INOff} \cdot C_F} = \frac{2 \cdot 2{,}5}{1 \cdot 10^{6} \cdot 120 \cdot 10^{-12}} = 41{,}6667 \cdot 10^{3} \qquad \textbf{(16)}$$

Formel 16: Berechnung des Widerstands R_{IN} für (GATE = LOW)

Diese Frequenz von 41,6667 kHz erlaubt eine einfache Auslegung des nachfolgenden Filters, da die Frequenz um den Faktor 100 unter der Mittenfrequenz des Modulators im normalen Betrieb liegt.

Der Schaltplan für das Modulatormodul ist in Abbildung 44 dargestellt.

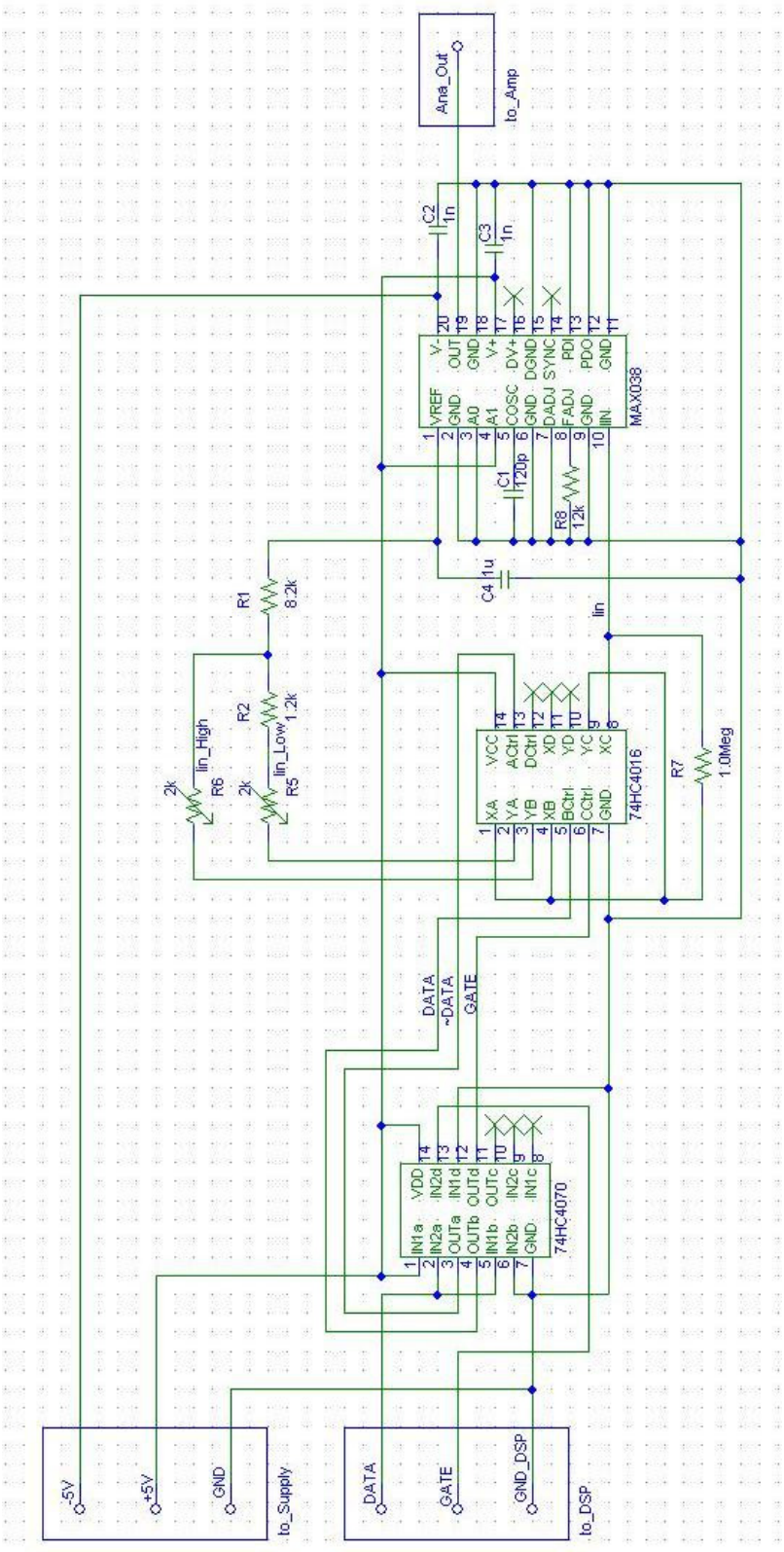

Abbildung 44: Schalplan Modul Modulator

7.5 Aufbau des Verstärkers / Filters

Um den Ausgang des MAX 038 vom Ausgang des Modulators zu entkoppeln und die Amplitude des Ausgangssignals anzuheben, wird dem MAX 038 ein zusätzlicher Operationsverstärker IC nachgeschaltet. Damit im Falle eines stumm geschalteten Ausgangs das Restsignal des MAX 038 unterdrückt wird, wird der Verstärker um Filtereigenschaften erweitert. Dabei sollen folgende Vorgaben erfüllt werden:

- ⇨ Erreichen des Durchlassbereichs ab einer Frequenz von höchsten 3,5 MHz
- ⇨ Möglichst gleiche Phasenlage im Bereich von 3,0 MHz bis 5,0 MHz
- ⇨ Verstärkungsfaktor von 2
- ⇨ Möglichst große Dämpfung des Restsignals des MAX 038 bei 50,0 kHz
- ⇨ 3dB Grenzfrequenz bei 500,0 kHz

Als Filterschaltung wird ein Hochpass zweiter Ordnung mit einfacher Mitkopplung gewählt. Die Schaltung entspricht dabei Bild 6-63 aus [13] (Seite 365). Dort sind ebenfalls die vereinfachten Berechnungsvorschriften zur Auslegung des Filters gegeben. Aus den Vorgaben folgt für die Elemente des Filters:

$$v_0 = 3 - \alpha = \frac{R_3}{R_4} + 1$$

$$v_0 = 3 - 0{,}9 = 2{,}1$$

$$\frac{R_3}{R_4} = 2{,}1 - 1{,}0 = 1{,}1$$

mit $R_3 = 1{,}2k\Omega$ folgt

$$R_4 = \frac{R_3}{1{,}1} = 1{,}09k\Omega \approx 1{,}0k\Omega \qquad \textbf{(17)}$$

Formel 17: Berechnung der Widerstände (Filter - R_3 und R_4)

Für die Kapazitäten gilt nach [13]:

$$C_1 = C_2 = C = \frac{1}{R \cdot 2 \cdot \pi \cdot f_g}$$

mit $R_1 = R_2 = R = 1{,}0k\Omega$ und $f_g = 500{,}0kHz$ folgt:

$$C = \frac{1}{1 \cdot 10^3 \cdot \Omega \cdot 2 \cdot \pi \cdot 500 \cdot 10^3\, Hz} = 318{,}3099\,pF \approx 330{,}0\,pF \qquad \textbf{(18)}$$

Formel 18: Berechnung der Widerstände (Filter – C_1 und C_2)

Die berechneten Bauteilwerte führen zu folgender Schaltung:

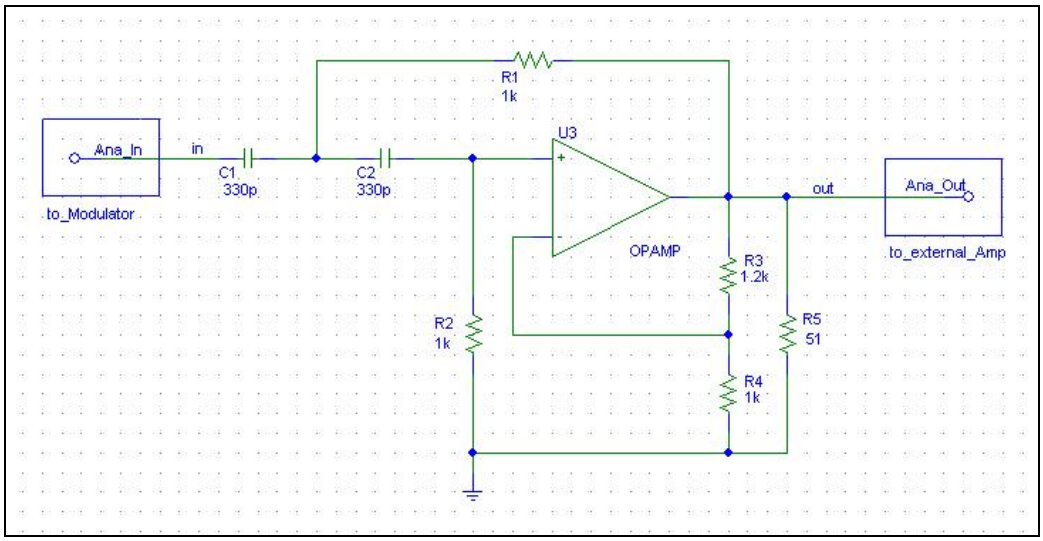

Abbildung 45: Schalplan Modul Verstärker / Filter

Bei einer Überprüfung der Filterauslegung über Pspice konnten folgender Frequenz- und Phasengang ermittelt werden.

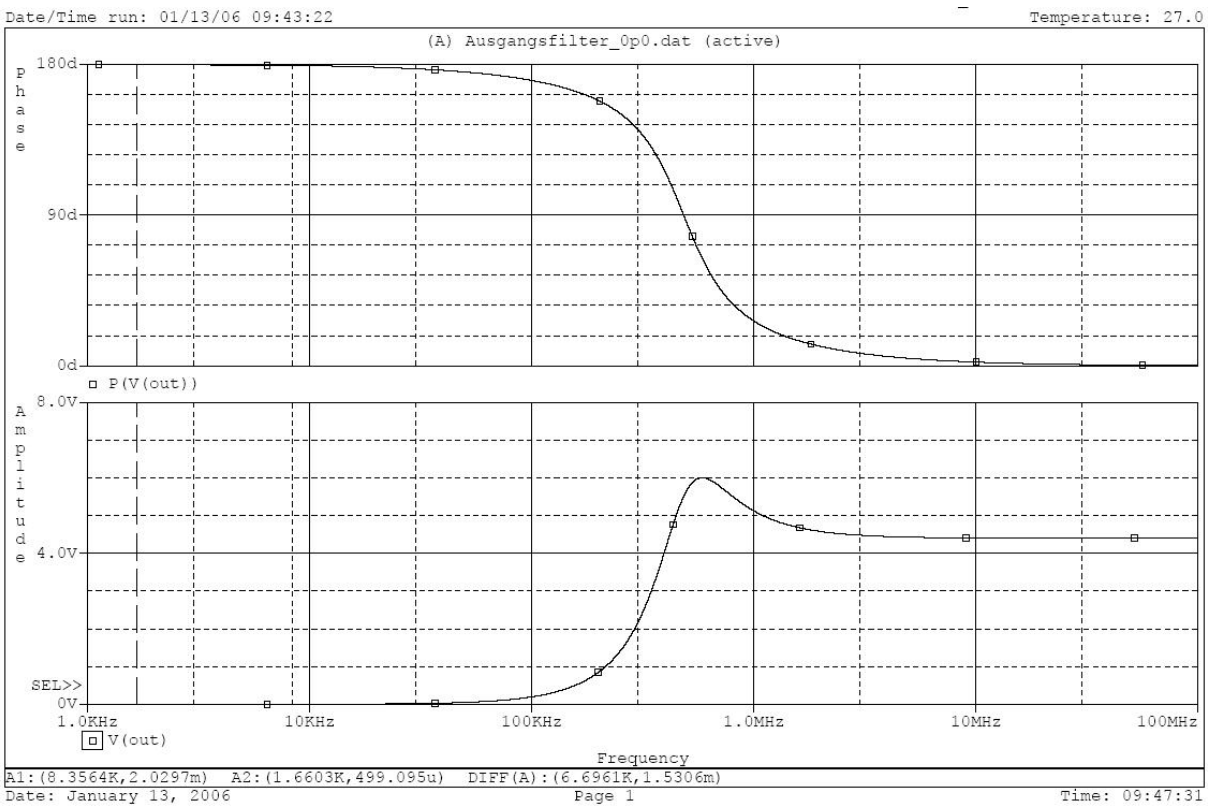

Abbildung 46: Frequenzgang Modul Verstärker / Filter

Allerdings spiegelt die Simulation nur das verhalten der Schaltung unter Verwendung eines idealen Operationsverstärkers wieder. Das reale verhalten dürfte davon abweichen. Um diese Abweichungen klein zu halten, fällt die Wahl für den verwendeten Operationsverstärker auf den IC LM6264N der Firma National Semiconductor, da dieser Operationsverstärker auch bei den hier verwendeten Frequenzen noch eine ausreichende Verstärkung von maximal 40dB bei annähernd linearer Phase erreicht. Das Verstärkungsbandbreitenprodukt liegt bei 175 MHz.

7.6 Aufbau der Spannungsversorgung

Zusätzlich zu den reinen Modulatoren wird der Schaltung eine Spannungsversorgung zur Erzeugung der notwendigen Spannungen hinzugefügt. Die Schaltung soll sowohl den Modulator und den Filter, als auch das DSP - Board versorgen. Dadurch kann die gesamte Schaltung, inklusive des DSP - Boards, in ein abgeschlossenes Gehäuse montiert werden.

Für die Erzeugung der Versorgungsspannungen wird ein DC / DC Wandler eingesetzt, der aus einer Eingangsspannung von +12 V die notwendigen Spannungen von +5 V und -5 V erzeugt. Um den DC / DC Wandler auslegen zu können, musste die Stromaufnahme der gesamten Schaltung im Betrieb gemessen werden. Sie beträgt:

Tabelle 45: Stromaufnahme der Schaltung

Bezeichnung	Stromaufnahme
DSP Board (normaler Betrieb)	320 mA
DSP Board (Telegramm senden)	300 mA
DSP Board (RESET Zustand)	170 mA
Modulator (einfach +5V Ruhe)	50 mA
Modulator (einfach -5V Ruhe)	50 mA
LED (7 * LED 20 mA)	140 mA

Aus diesen Werten ergeben sich folgende Werte für die Stromaufnahme der gesamten Schaltung:

$$
\begin{aligned}
I_{Max+5V} &= I_{MaxDSP} + I_{MaxMod} + I_{MaxLED+5V} = 320mA + 100mA + 140mA = 560mA \\
I_{Min+5V} &= I_{MinDSP} + I_{MinMod} + I_{MinLED+5V} = 170mA + 100mA + 40mA = 310mA \\
I_{Max-5V} &= I_{MaxMod} + I_{MaxLED-5V} = 100mA + 20mA = 120mA \\
I_{Min+5V} &= I_{MaxMod} = 100mA
\end{aligned}
\qquad \textbf{(19)}
$$

Formel 19: Berechnung der min. und max. Stromaufnahme des Modulators

Um in der Schaltung Reserve zu haben, fällt die Entscheidung für den DC / DC Wandler auf einen integrierten IC TEN8-1221 der Firma Traco Power. Dieser DC / DC Wandler liefert bei einer Eingangsspannung von +9 V bis +18 V und einer Ausgangsspannung von +5 V / -5 V einen maximalen Ausgangsstrom von 800 mA, was für die Versorgung der Schaltung ausreichend ist.

Hierbei muss der minimale Ausgangsstrom des Wandlers beachtet werden, ab dem er zuverlässig arbeiten kann. Er beträgt beim TEN8-1221 10 % des maximalen Ausgangsstroms. In unserem Fall ist I_{OutMin} 80 mA. Dieser Strom wird auf der +5 V Seite immer allein durch das DSP - Board sicher erreicht. Auf der -5 V Seite beträgt der minimale Laststrom nur 100 mA. Deshalb wird hier eine zusätzliche Last vorgesehen, um den Wandler auch dann noch stabil betreiben zu können, wenn einzelne Komponenten des Modulators ausfallen sollten.

Neben dem DC / DC Wandler wird ein zusätzlicher Treiber IC in die Schaltung integriert, der den Zustand der Schaltung über LEDs für den Anwender sichtbar machen soll. Dazu sind sieben Statusanzeigen vorgesehen. Diese sind:

- ⇨ RESET aktiv
- ⇨ Datensignal für Balisenkanal 1
- ⇨ Gatesignal für Balisenkanal 1
- ⇨ Datensignal für Balisenkanal 2
- ⇨ Gatesignal für Balisenkanal 2
- ⇨ +5 V Versorgung vorhanden
- ⇨ -5 V Versorgung vorhanden

Insgesamt wird die Platine so ausgelegt, dass sie direkt auf das bestehende DSP - Board montiert werden kann. Alle Signalleitungen und Verbindungen sollen über Steckkontakte mit den Quellen verbunden werden, um einen modularen Aufbau zu erhalten und das DSP - Board nicht beschädigen zu müssen. Die Spezifikation für Erweiterungskarten für das DSP - Board ist in [10] (Abbildung 1, Seite 4) gegeben.

Der Schaltplan ist in Anhang 11.3 und das vorläufige Layout der gesamten Platine in Anhang 11.4 zu finden.

7.7 Ergebnisse

Das Ergebnis des Modulatoraufbaus ist eine zweilagige Platine, auf der zwei FSK – Modulatoren und die Spannungsversorgung für DSP – Borad und Modulator implementiert sind. Alle Verbindungen zum Gehäuse oder dem DSP – Board sind als Steck- oder Schraubverbindungen ausgeführt, um keine Komponenten nachhaltig beschädigen zu müssen. Abbildung 47 zeigt die voll bestückte Platine auf dem DSP – Board montiert.

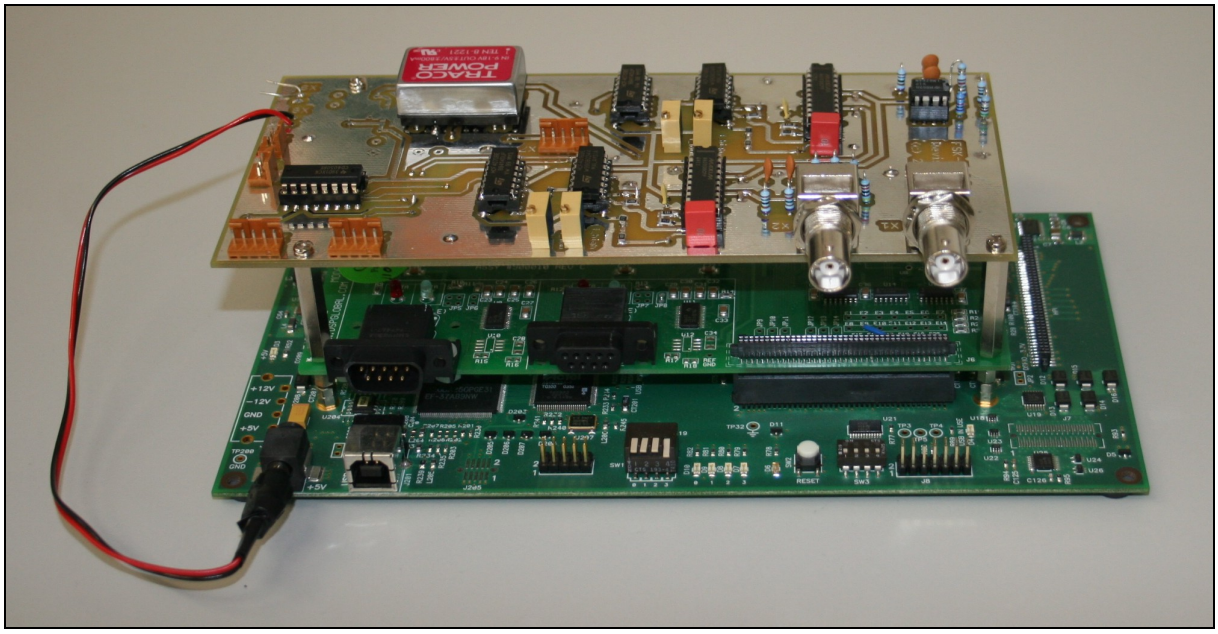

Abbildung 47: FSK – Modulator (bestückte Platine)

Bei Tests mit der Platine mit verschiedenen Signalen, die über den Diagnosekanal des DSP – Systems erzeugt werden, kann die Leistungsfähigkeit und Funktion der Modulatorplatine getestet werden. Hierzu werden die analogen FSK – Signale sowohl im Ruhezustand, als auch bei Ausgabe einer logischen 1, einer logischen 0 und einem Wechsel aufgenommen.

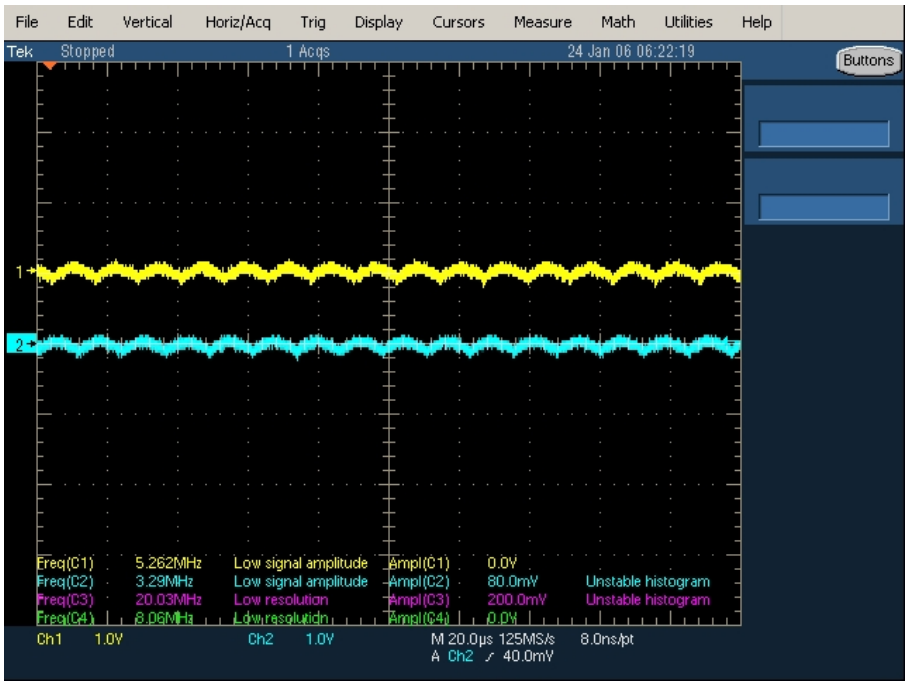

Abbildung 48: FSK – Modulator (GATE = LOW; DATA = X)

Abbildung 48 zeigt das Ausgangssignal des FSK – Modulators im Ruhezustand. Hier ist zu sehen, dass der Ausgangsfilter das Restsignal des MAX 038 sehr gut wegfiltert. Der Signalpegel wird hier auf 320 mV Spitze / Spitze gesenkt.

Bei Ausgabe einer logischen 0 oder 1 beträgt der Pegel 2,8 V Spitze / Spitze. Die Dämpfung des Signals vom „Senden“ Fall zum Ruhezustand beträgt demnach 18,84 dB. Dabei treten im Ausgangssignal keine größeren Verzerrungen auf, wie in Abbildung 49 und Abbildung 50 zu sehen ist.

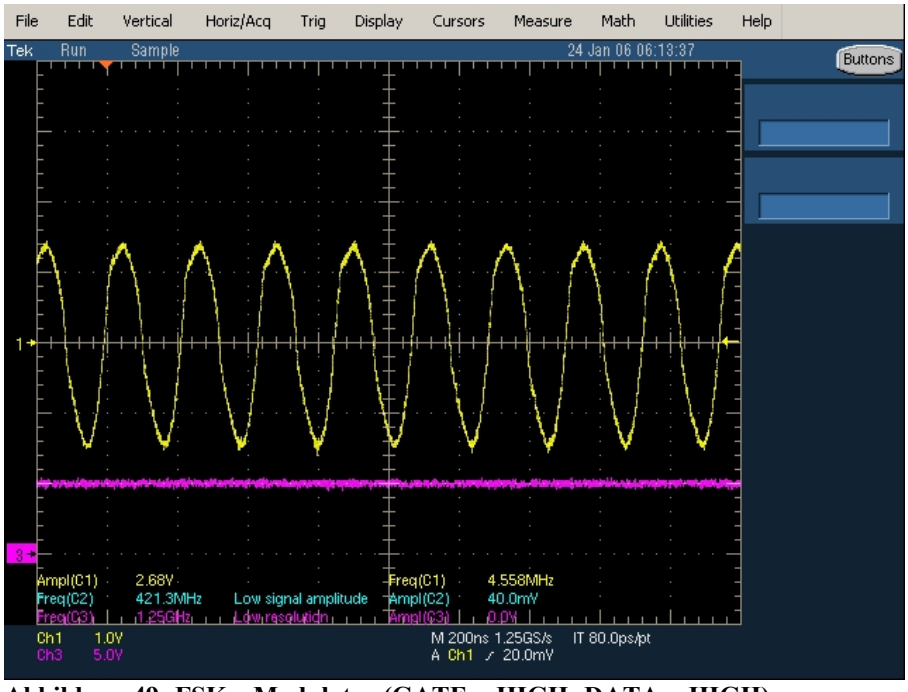

Abbildung 49: FSK – Modulator (GATE = HIGH; DATA = HIGH)

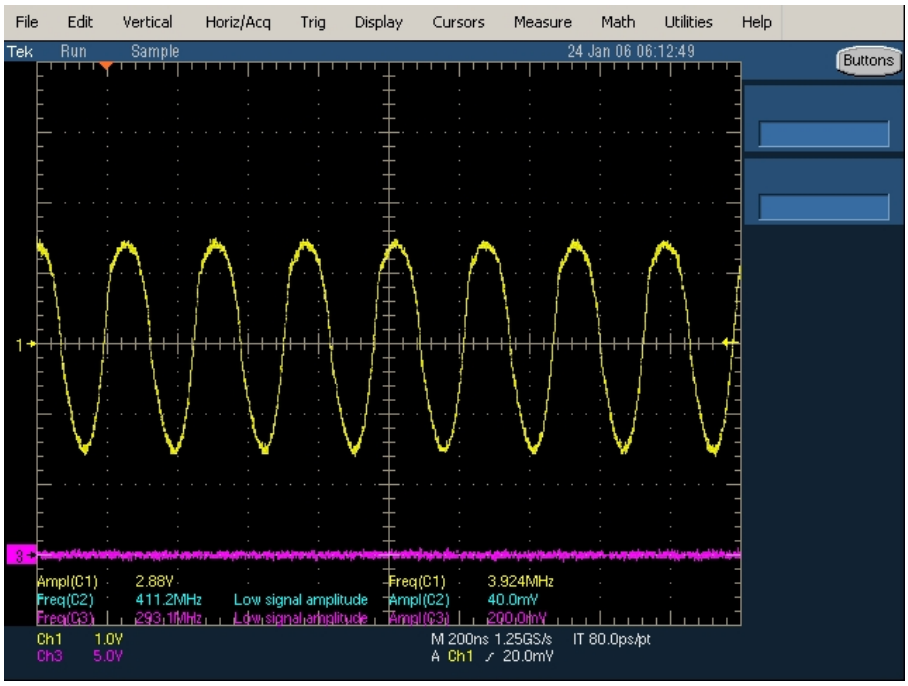

Abbildung 50: FSK – Modulator (GATE = HIGH; DATA = LOW)

Abbildung 51 zeigt schließlich das Ausgangssignal bei einem Wechsel zwischen logischen 0 und 1 Pegel. Wie dort zu erkennen ist, wird die Anforderung nach Phasenlinearität des Ausgangssignals eingehalten.

Einziger Kritikpunkt könnte der leichte Unterschied in der Amplitude zwischen dem Senden einer logischen 1 und 0 sein. Dieser Unterschied rührt von dem Frequenzgang des Ausgangsfilters her. Er ist aber mit wenigen Millivolt zu vernachlässigen.

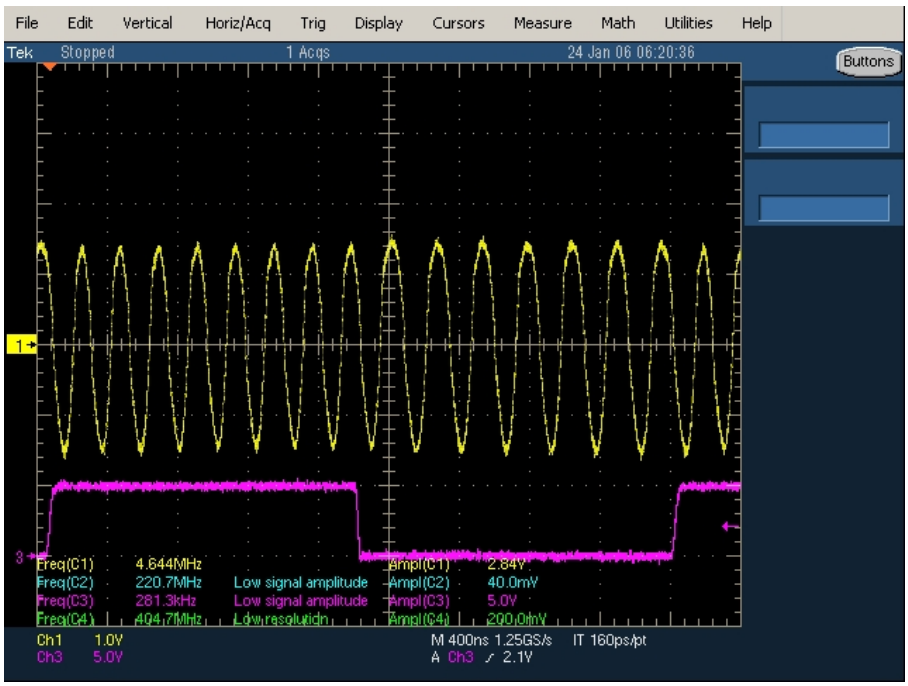

Abbildung 51: FSK – Modulator (GATE = HIGH; DATA = LOW to HIGH)

8 Zusammenfassung und Ausblick

Das Ergebnis dieser Arbeit ist ein funktionsfähiges Gerät zur Anbindung eines realen Fahrzeugrechners and das RailSiTe - Labor. Die Software und anschließende Modulation machen es möglich, das Fahrzeug auf der simulierten Strecke mit bis zu 320 km/h zu bewegen. Ein Fahrzeugrechner des Typs TCC der Firma Siemens hat in Tests mit der Balisenansteuerung problemlos zusammengearbeitet.

Abbildung 52: Gerät zur Balisenübertragung

Die Kodierung der Balisentelegramme ist über den DSP zwar möglich, allerdings ist es trotz starker Optimierungen nicht gelungen, die Kodierzeit der Telegramme auf ein Niveau zu bringen, dass auf einem handelsüblichen PC - System erreicht werden kann. Die Kodierung der Balisentelegramme sollte daher auf Grund der Echtzeitanforderungen des DSP Systems vor der Übertragung der Telegrammdaten vom RailSiTe - Labor zum DSP im BTM des RailSiTe - Labors durchgeführt werden. Dieser Punkt wird umso wichtiger, wenn zwei Balisenkanäle über das DSP - System angesprochen werden sollen. Für diesen Fall ist das System während der Kodierung der Telegramme ausgelastet und der Fahrzeugrechner für den anderen Kanal nicht mehr erreichbar.

Die Modulatorschaltung hat sich im Betriebt bewährt und arbeitet zuverlässig mit dem Fahrzeugrechner zusammen. Das ist insbesondere deshalb erfreulich, da es mit dem alten Modulatoraufbau mit harter Frequenzumtastung zu Problemen in der Kommunikation mit dem Fahrzeugrechner gekommen ist.

Es bleibt zu überlegen, ob der zweite Balisenkanal des DSP - Systems statt zum Anschluss eines zweiten Fahrzeugrechners für die Simulation eines Euroloop Systems genutzt werden sollte. Damit könnte der volle Umfang der PZB und LZB des ETCS Level 1 simuliert werden.

Alternativ könnte der zweite Balisenkanal auch zur Simulation von physikalischen Übertragungsfehlern, wie z.B. Übersprechen eines zweiten Kanals in eine Balisenübertragung genutzt werden. Solche Fehler sind allerdings bei der Balisenübertragung eher unwahrscheinlich, da die Übertragungsstrecke zwischen Zug und Balise nur wenige Zentimeter beträgt und die Übertragung der Balisen durch den Zug selbst bei Überfahrt angeregt wird. Durch Einhaltung von Sicherheitsabständen kann also ein Übersprechen zweier Balisen ausgeschlossen werden. Gegen externe Störeinflüsse ist die Übertragung auf Grund der geringen Distanz und der Ein-

bettung der Balisen in das Gleisbett weitestgehend abgeschirmt. Daher ist eine Nutzung des zweiten Balisenkanals als Schnittstelle zu einem zweiten Fahrzeuggerät sinnvoller.

9 Literaturverzeichnis

Tabelle 46: Literaturverzeichnis

Index	Quellen
[1]	„Form Fit Function Specification – Coding Strategy“; Issue: 3.0.0; Eurosig consortium; 28.02.1997
[2]	„Interface- und Telegrammspezifikation für die Kommunikation zwischen RailSiTe und Balisenansteuerung“; DLR - Institut für Verkehrsführung und Fahrzeugsteuerung; Dipl. Ing. Volker Knollmann; 09.03.2005
[3]	„Skript Informations- und Kodierungstheorie“; Fachhochschule BS/WF; Dr. Frank Jäger; Wintersemester 2004/2005
[4]	„Elektrotechnik Tabellen Energieelektrotechnik“; Westermann Schulbuchverlag GmbH; Brechmann, Dzieia,…; 5. Auflage 2002
[5]	„Formeln und Hilfen zur höheren Mathematik“; Binomi Verlag; Gerhard Merziger, Günter Mühlbach, …; 3. Auflage 06.10.1999
[6]	„Datenblatt TMS320C6713, TMS320C6713B“; Revision Mai 2004; Texas Instruments Online Datenbank (http://www.ti.com) sprs1861.pdf; Dezember 2001
[7]	„TECHNICAL REFERENCE MANUAL RS-232-C AND RS-422 SERIAL COMMUNICATION PORT EXPANSION DAUGHTER CARDS“; 02.06.2005; DSP Global (http://www.dspglobal.com) Int_Card_Manual.pdf
[8]	„Datenblatt MAXIM MAX038“; 19-0266 Revision 2; Maxim Online Datenbank (http://www.maxim-ic.com) MAX038.pdf
[9]	„Diplomarbeit: Architekturüberlegungen zu einem Simulatorsystem für Interoperabilitätstests an ERTMS/ETCS-Komponenten“; Technische Universität Carolo-Wilhelmina zu Braunschweig, Jörg Sump, Dezember 2002
[10]	„TMS320 Cross-Platform Daughtercard Specification“; Revision 1.0; Texas Instruments Online Datenbank (http://www.ti.com) spra711.pdf
[11]	„Form Fit Function Specification - Eurobalise“; Issue: 2.2.1; Eurosig consortium; 12.09.2003; (www.aeif.org) SUBSET-036-v221.pdf
[12]	„ERTMS / ETCS Class 1 – System Requirement Specification Chapter 7 – ERTMS / ETCS language“ ; Issue: 2.2.2; Eurosig consortium ; 02.02.2001; ; (www.aeif.org) SUBSET-026-v222.pdf
[13]	„Elektronik für Ingenieure“; Springer verlag; Hering, Bressler, Gutekunst; 4. Auflage 2001

10 Abkürzungen und Begriffsbedeutungen

Tabelle 47: Abkürzungen und Begriffsbedeutungen

Symbol	Bedeutung
Balise (Eurobalise)	Modul zur Zug- / Streckekommunikation, das fest in das Gleisbett eingebettet wird und bei Überfahrt eines Zuges aktiviert wird. Dabei sendet es zyklisch ein Telegramm an den überfahrenden Zug.
Bitslip	(engl. slip = durchschlüpfen) beschreibt hier den Vorgang, das der Sender beim Versenden einer Bitfolge ein einzelnes Bit auslässt, oder der Empfänger ein einzelnes Bit einer Bitfolge nicht detektiert.
Bitinsertion	(engl. insert = einfügen) beschreibt hier den Vorgang, das der Sender beim Versenden einer Bitfolge ein einzelnes Bit einfügt, oder der Empfänger ein einzelnes Bit einer Bitfolge doppelt detektiert, oder ein nicht vorhandenes Bit erkennt.
BTM	Ein „**B**alise **T**ransmission **M**odul“ ist das Modul des RailSiTe Labors, dass die Balisentelegramme an den DSP sendet und den DSP zum Senden der Telegramme über die HF Schnittstelle scharf macht. Innerhalb des ETCS Systems ist das BTM die Einrichtung, die streckenseitig für die Ansteuerung der Balisen zuständig ist.
DLR	„**D**eutsches Zentrum für **L**uft- und **R**aumfahrt“. Forschungseinrichtung für Luft- und Raumfahrt, aber auch Automobil und Verkehr.
DMI	„**D**river **M**achine **I**nterface“ Für ETCS einheitliches Interface zum Zugriff auf die Funktionen des EVC. Die Ausgabe erfolgt über einen Touchscreen.
DSP	Mit „**D**igital **S**ignal **P**rocessor“ ist in diesem Fall das gesamte DSK6713 Evaluationsboard samt seiner aktuellen Software gemeint, das die Schnittstelle zwischen dem **BTM** und der eigentlichen Antenne zur Funkübertragung bildet.
Eurosig Konsortium	Zusammenschluss von europäischen Firmen mit dem Ziel der Definition einer einheitlichen europäischen Zugleit- und Sicherheitstechnik (ETCS).
ESB	**E**xtra **S**haping **B**its, werden bei der Kodierung zur Berechnung der Checksumme benutzt
ETCS	„**E**uropean **T**rain **C**ontrol **S**ystem“. Einheitliches europäisches System zur Zugbeeinflussung, Zugbeeinflussung und Steuerung. Wir durch das Eurosig Konsortium erarbeitet und spezifiziert.
EVC	„**E**uropean **V**ital **C**omputer“. Zentraler Fahrzeugrechner des ETCS Systems.
FSK	„Frequency Shift Keying“ Frequenzmodulationstyp, bei dem zwischen zwei Frequenzen, passend zu den logischen Zuständen des Modulationssignals, umgetastet wird.
GSM-R	**„G**lobal **S**ystem for **M**obile Communications – **R**ailway“. Das GSM-R Netz ist ein Unternetz des GSM Netzes, das speziell für die Telekommunikation von Komponenten der Zugleit- und Sicherungstechnik reserviert ist.
IC	„**I**ntegrated **C**ircuit“ integrierte Schaltung, gekapselt in einem einzigen Baustein
ISR	„**I**nterupt **S**ervice **R**outine“ Funktion, die in einer Software nach dem Auftauchen eines Interrupts zur Bearbeitung dieses Ereignisses aufgerufen wird.
LEU	„**L**inside **E**lectronic **U**nit“ Diese Einheit wird an der Strecke montiert und steuert angeschlossene Eurobalisen oder Euroloops je nach Status des angeschlossenen Signals

LZB	„**l**inienförmige **Z**ug**b**eeinflussung“ Die LZB bietet eine kontinuirliche Datenübertragung von der Strecke zum Fahrzeug. Dadurch ist das Fahrzeug immer über die aktuellen Streckenverhältnisse infomiert und kann schneller und besser auf Veränderungen reagieren. Deshalb wird eine LZB meist auf Strecken mit starker Auslastung, oder hohen Geschwindigkeiten verwendet. Unter ETCS wird die LZB hauptsächlich durch das GSM-R System sichergestellt. Als Rückfallebene kann auch der Eruoloop benutzt werden, der stark den heute in Deutschland verwendeten Linienleitern ähnelt.
PZB	„**p**unkförmige **Z**ug**b**eieinflussung“ Die PZB wird in der Zugleit- und Sicherungstechnik zur punktförmigen Datenübertragung von der Strecke an das Fahrzeug genutzt. Gängige Vertreter sind Indusi und PZB 80 in Deutschland. Sie wird meist auf langsamen, oder schwach ausgenutzten Strecken benutzt. Im Gegensatz zu diesen beiden Systemen ist die Eurobalise in ETCS in der Lage, große Datentelegramme an das Fahrzeug zu übertragen.
RailSiTe - Laboratory	„**Rail**way **Si**mulation and **Te**sting“ Labroratory. Labor des DLR zum funktionalem Test von Komponenten der Bahn Leit-, Betriebs- und Sicherheitstechnik mit speziellem Schwerpunkt auf ETCS.
RBC	„**R**adio **B**lock **C**enter“ Das Radio Block Center ist die Einrichtung des ETCS Systems, das in den ETCS Leveln 2 und 3 alle Daten über das GSM-R Netz sendet und empfängt.
SB	„**S**crambling **B**its“, werden bei der Kodierung zur Verwürfelung der Nutzdaten benutzt
UART	engl. „**U**niversal **A**synchronous **R**eceiver **T**ransmitter“, hier ist ein Peripheriebaustein gemeint, der die Anbindung des DSP an externe Geräte mittels einer RS 232 Schnittstelle ermöglicht. Dieser Baustein befindet sich auf der Daughtercard, die and das DSP Board angeschlossen ist (siehe Kapitel 3.2.1).

11 Anhang

11.1 Erweiterung DSP Firmware auf mehrere RailSiTe Kanäle

Die bisherige DSP - Software unterstützt nur das Senden von Telegrammen über einen Balisenkanal. Da der DSP grundsätzlich die Möglichkeit bietet, über zwei Balisenkanäle gleichzeitig Daten zu senden, soll die DSP - Software dahingehend geändert werden, dass zwei RailSiTe - Kanäle gleichzeitig vom DSP behandelt werden können.

Wie beim Diagnosekanal stellt sich auch hier das Problem, dass die DSP - Software in der jetzigen Version kein Multitasking unterstützt. Es ist also nicht möglich, die DSP - Software einfach zweifach parallel auf dem DSP auszuführen. Eine Änderung der Software hierhin gehend wäre aber auch zu aufwendig, da dann zusätzlich ein Betriebssystem implementiert werden müsste, dass diese parallele Ausführung von Prozessen unterstützt.

Aus diesen Gründen wird eine einfachere Lösung angedacht, bei der der DSP zwar weiterhin immer nur das Protokoll für einen Balisenkanal bearbeiten kann, allerdings auf der Empfangsseite zum RailSiTe - Labor für alle Kanäle jederzeit ansprechbar ist.

Sollte der DSP belegt sein, werden ankommende RailSiTe - Pakete, wie bei aktiviertem Diagnosekanal auch, mit einem Acknowledge quittiert, in dem dem RailSiTe mitgeteilt wird, dass das aktuelle Paket ignoriert wird.

Die Implementation solch eines Konstrukts erfordert zusätzliche Strukturen, um eine Schnittstelle zwischen der UART Interrupt Service Routine (UART ISR) und dem RailSiTe - Protokollhandler zu schaffen. Dabei wird die komplette Intelligenz, die zum Empfang eines RailSiTe - Pakets notwendig ist, in die ISR verlagert. Sie übergibt nur dann ein Datenpaket an den Protokollhandler, wenn ein komplettes, fehlerfreies Paket empfangen wurde.

Die neue Datenstruktur hat folgenden Aufbau:

```
typedef struct
{
      uint8     data_aru8[RS232_PAKET_MAX_SIZE];      // array for paket data
      tlgHeader header_s;                             // structure for telegram header
}t_RS232Paket_buffer;

typedef struct
{
      t_RS232Paket_buffer buffer_s[2];    // array for paket data
      uint16    byte_cnt_u16;             // counter for received      bytes in UART ISR
      uint8     isr_state_u8;             // state of UART ISR for actual port
      uint8     isr_index_u8;             // index on actual buffer for UART ISR
      uint8     prot_index_u8;            // index on actual buffer for RailSiTe protocol handler
      uint8     rec_ready_u8;             // true, if complete packet was received
      uint8     rec_start_u8;             // true, when first byte of packet was received
      uint8     rec_timeout_u8;           // true, if timeout occured in receiving of paket
      uint16    rec_timeout_cnt_u16;      // counter for reception timeout
}t_RailSiTe_port;
```

```
typedef struct
{
      t_RailSiTe_port port_s[NUM_RAILSITE_PORTS];   // array for packet data
      uint8     dsp_busy_u8;                        // true, if dsp is busy handling a packet
}t_RailSiTe_prot;
```

Die Struktur **t_RailSiTe_prot** enthält für jede der beiden RailSiTe - Kanäle einen doppelt ausgelegten Paketpuffer. Dadurch kann auch dann ein Paket durch die UART ISR empfangen werden, wenn der DSP im Protokollhandler ein Paket für denselben Kanal auswertet. In diesem Fall wird das neue Paket ignoriert.

Zusätzlich enthält die Struktur für beide Schnittstellen mehrere Flags, die für die Kommunikation zwischen der UART ISR und dem Protokollhandler notwendig sind. Diese Flags sollen nachfolgend kurz beschrieben werden, da ihre Bedeutung für das Verständnis der neuen Softwarestruktur unabdingbar ist.

Tabelle 48: Struktur t_RailSiTe_prot

Name	Zugriff		Beschreibung
	Prot.	ISR	
byte_cnt_u16	-	R/W	Diese Variable enthält die Anzahl der bereits von einem RailSiTe Paket empfangenen Bytes. Er wird ausschließlich in der UART ISR verwendet. Dort kann er nicht lokal gespeichert werden, da die UART ISR zwar zyklisch aufgerufen wird, aber alle lokalen Variablen nach Ende der Funktion verloren gehen.
isr_state_u8	-	R/W	In dieser Variablen wird der aktuelle State gespeichert, in der sich die Empfangsstatemachine in der UART ISR für den jeweiligen RailSiTe Kanal befindet. Der Protokollhandler greift nicht auf diese Variable zu.
isr_index_u8	-	R/W	Der Index zeigt immer auf den Puffer, der von der UART ISR benutzt werden soll. Er enthält immer den gegenteiligen Wert der Variable **prot_index_u8.** Speichert also die UART ISR gerade ein neues Paket in Puffer 0, so greift der Protokollhandler ausschließlich auf den Puffer 1 zu. Die Umschaltung erfolgt nur in der UART ISR, wenn ein komplettes Paket empfangen wurde und der Protokollhandler kein Paket auswertet (**dsp_busy_u8** = 0).
prot_index_u8	R	R/W	Der Index zeigt immer auf den Puffer, der vom Protokollhandler benutzt werden soll.
rec_ready_u8	R/W	R/W	Dieses Flag wird von der UART ISR immer dann gesetzt, wenn ein komplettes Paket empfangen wurde, dass dem Protokollhandler zur Auswertung übergeben werden soll. Dieser löscht das Flag, wenn er die Auswertung des Pakets begonnen hat. Keiner der beiden Funktionen darf dieses Flag setzen UND löschen.

	Zugriff		
rec_start_u8	-	R/W	Dieses Flag wird von der UART ISR gesetzt, wenn der Empfang eines neuen Pakets begonnen hat. Es dient dazu, die Empfangsdauer eines Pakets zu überwachen, um gegebenenfalls einen Timeout zu erzeugen, falls ein angefangenes Telegramm nicht komplett empfangen wird.
rec_timeout_u8	-	R	Dieses Flag wird in der Funktion **mainLoop (…)** immer dann gesetzt, wenn ein Timeout beim Empfang eines Pakets aufgetreten ist. Die UART ISR liest dieses Flag und setzt sich selbst zurück. Dabei wird auch dieses Flag wieder gelöscht. Die Timeoutüberwachung kann weder im Protokollhandler, noch in der UART ISR intern durchgeführt werden, da beide Funktionen nicht in definierten zeitlichen Abständen aufgerufen werden.
rec_timeout_cnt_u16	-	-	Dieser Zähler wird in der Funktion **Timer0_ISR (…)** im Millisekundentakt inkrementiert, wenn das Flag **rec_start_u8** den Empfang eines neuen Pakets anzeigt. Anschließend wird es in der Funktion **mainLoop (…)** überwacht, um einen Timeout feststellen zu können.

R: Lesezugriff
W: Schreibzugriff

Das Flag **dsp_busy_u8** ist nur einfach vorhanden. Es wird immer dann vom Protokollhandler gesetzt, wenn der DSP gerade mit der Auswertung eines RailSiTe - Pakets beschäftigt ist. Erst nach Ende der Verarbeitung des Pakets wird dieses Flag vom Protokollhandler wieder gelöscht.

Dadurch kann die UART ISR alle Pakete, die während dieses Zeitraumes empfangen werden, mit einem Acknowledge bestätigen und ignorieren. Der DSP ist also nicht für Anfragen des RailSiTe blockiert.

Das Zusammenspiel aller Flags im laufenden Betrieb ist kompliziert und soll in den nachfolgenden Grafiken veranschaulicht werden. Grundsätzlich sollten durch diese Struktur beliebig viele RailSiTe - Kanäle unterstützt werden. Allerdings geht bei mehreren Kanälen die Echtzeitfähigkeit der DSP Software verloren, da immer nur ein Paket zur Zeit ausgewertet werden kann.

Während dieser Zeit werden alle ankommenden Pakete zwar quittiert, aber ignoriert. Je mehr Kanäle unterstützt werden, desto öfter wird dieser Fall auftreten. Das ist vor allem für die Pakete problematisch, in denen Sequenzen gestartet werden sollen, das sie meist punktgenau vom RailSiTe - Labor an den DSP gesendet werden, um die Balisentelegramme genau zum richtigen Zeitpunkt über die HF - Schnittstelle zu senden.

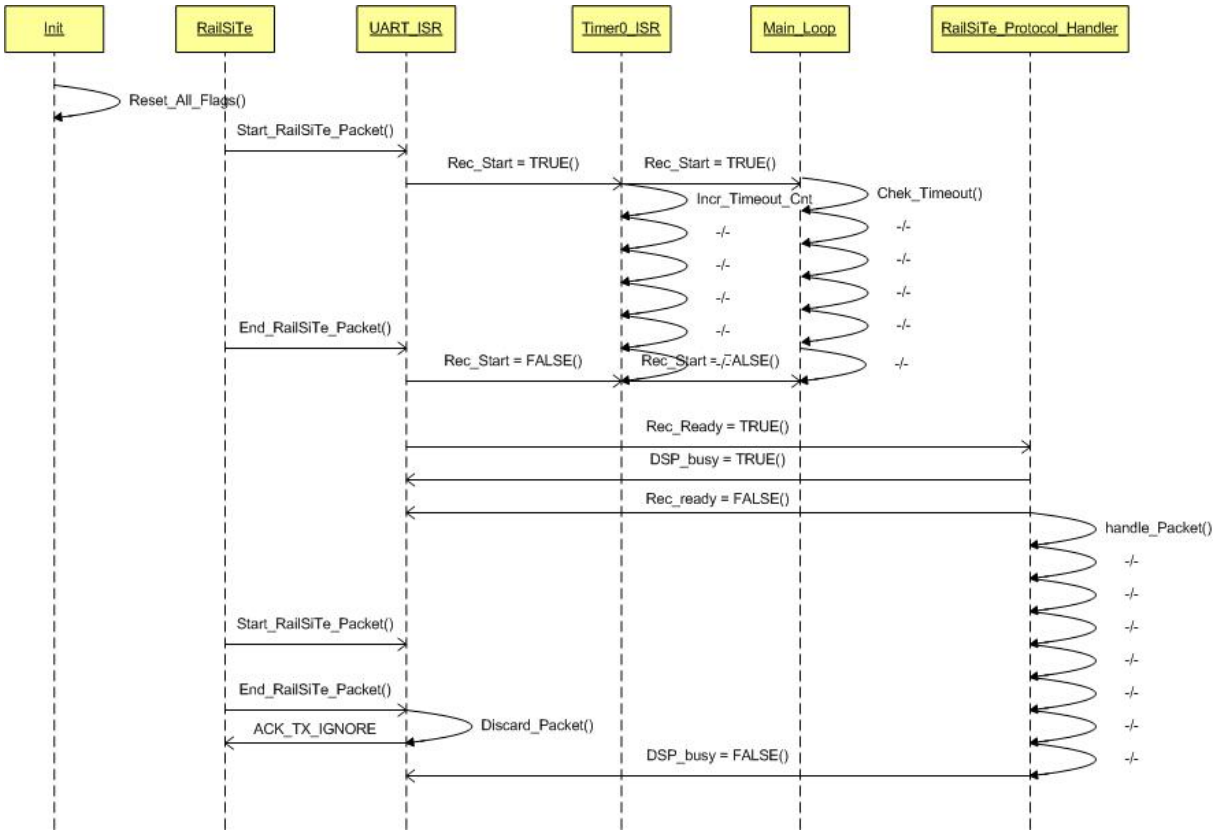

Abbildung 53: Sequenzdiagramm Empfang RailSiTe Paket mit DSP-Busy

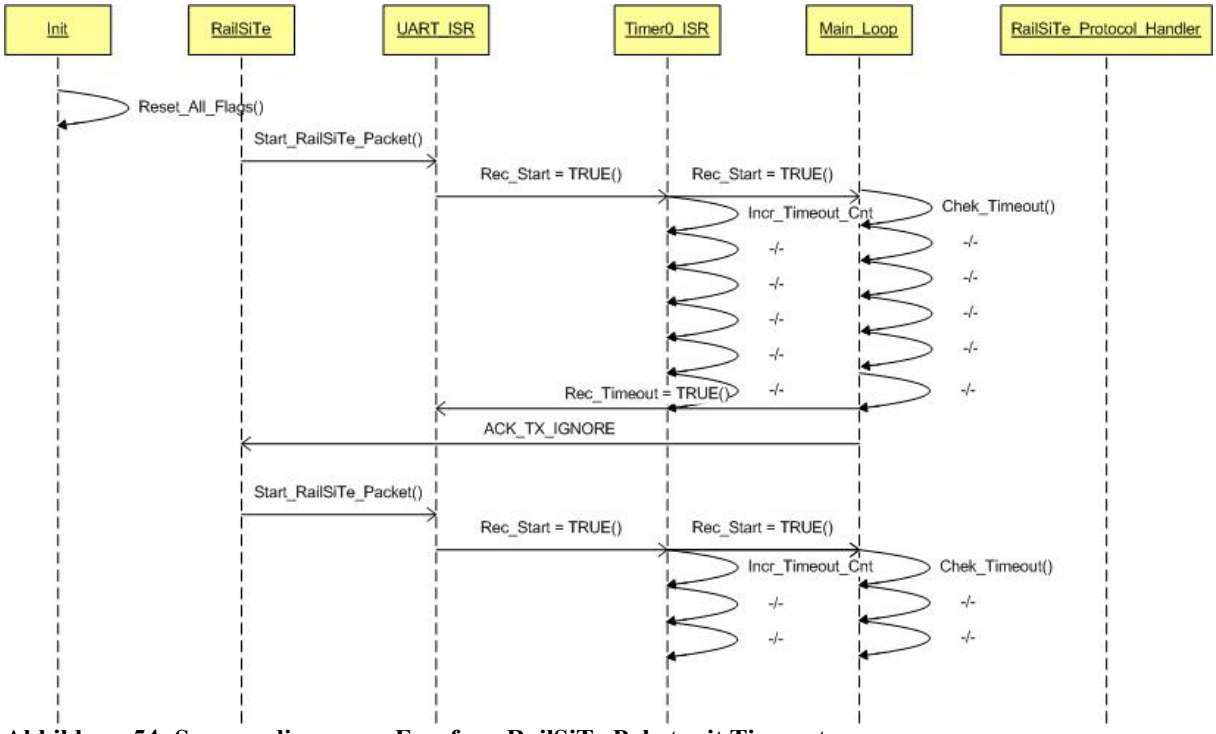

Abbildung 54: Sequenzdiagramm Empfang RailSiTe Paket mit Timeout

11.2 Messung und Korrektur der Systemlatenzzeiten

Das DSP System soll zur Anbindung des realen Fahrzeuggerätes Telegramme in einem fest definierten Zeitfenster senden. Die Einhaltung dieser Zeitfenster ist sehr wichtig, da die Fahrzeuggeräte selbst alle Zeiten überprüfen und Telegramme für ungültig erklären, wenn nicht alle zeitlichen Beschränkungen eingehalten werden.

Zu diesem Zweck übermittelt das BTM des RailSiTe - Labors alle notwendigen Balisentelegrammdaten über die Pakete **RS2DSP_StoreTlg** and das DSP - System. Anschließend gibt das BTM einen festen zeitlichen Ablauf vor, in dem die einzelnen Balisentelegramme vom DSP an das Fahrzeuggerät gesendet werden sollen. Dieser Ablauf wird als „Sequenz“ bezeichnet und über das Paket **RS2DSP_DefAndRunSeq** im DSP gespeichert und anschließend sofort gestartet. Eine solche Sequenz besteht für jedes Balisentelegramm, dass innerhalb der Sequenz gesendet werden soll, aus drei Komponenten. Diese sind:

- ⇨ **T_{Pause}:** Zeit vom Ende des letzten Telegramms bis zum Start des neuen Telegramms.
- ⇨ **T_{Senden}:** Dauer des Balisentelegramms
- ⇨ **TlgSlot:** Nummer des Telegrammspeicherplatzes, der verwendet werden soll

Das BTM des RailSiTe - Labors, dass die Verbindung des DSP - Systems zum RailSiTe - Labor darstellt und dieses mit Daten versorgt, bezieht seine Sequenzdefinition dabei immer auf den Zeitpunkt des Sendens des RS 232 Pakets, das den DSP zum Senden der Telegramme an das Fahrzeuggerät innerhalb einer Sequenz veranlasst.

Da das DSP - System nur über eine begrenzte Rechenleistung verfügt, kann es die Daten des RailSiTe - Labors nicht in Nullzeit an die Fahrzeuggeräte weiterreichen. Zwischen dem Empfang eines Balisentelegramms vom RailSiTe - Labor und dem Start der Sequenz zum Fahrzeuggerät liegt eine Latenzzeit, die hier mit T_{LatDSP} bezeichnet werden soll. Diese Zeit ist in den Sequenzvorgaben des BTM nicht berücksichtigt. Abbildung 55 soll diesen zeitlichen Ablauf anschaulich erklären.

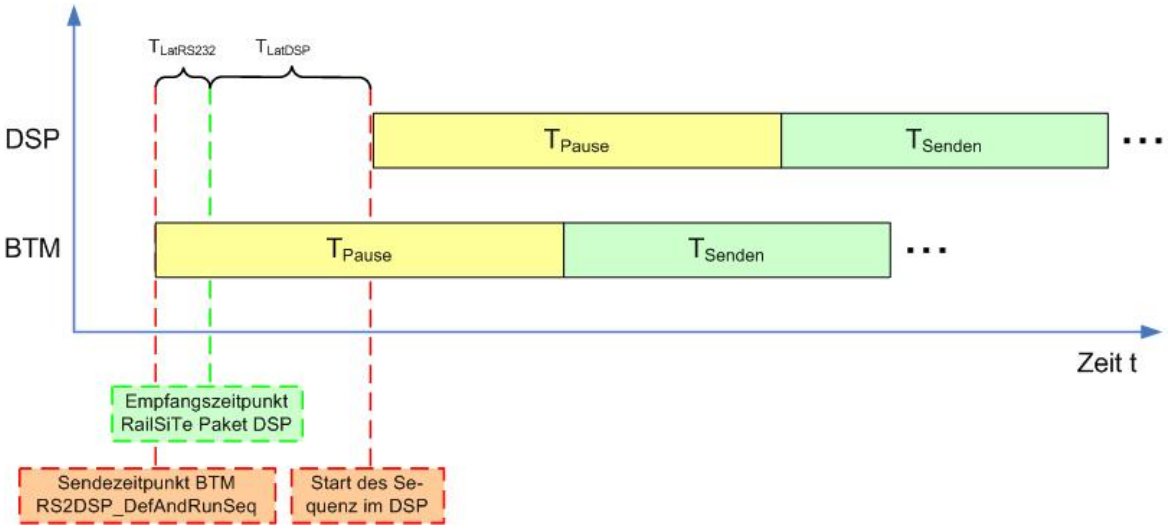

Abbildung 55: Latenzzeiten des DSP - Systems

Um festzustellen, ob diese Latenz zu Problemen in der Kommunikation mit dem Fahrzeuggerät führen kann, sollen in diesem Kapitel die Latenzzeiten für kurze und lange Telegramme gemessen und die Ergebnisse bewertet werden.

Die nachfolgende Tabelle gibt einen Überblick über die Latenzzeiten des DSP - Systems. Zum Test wurden lange und kurze Telegramme mit beliebigem Inhalt verwendet. Die Sequenzen und das in ihnen verwendete Balisentelegramm sind in den folgenden RailSiTe - Paketen gespeichert.

Tabelle 49: kodiertes Balisentelegramm (lang – Test Latenzzeit)

Byte Nr.	Daten															
0 – 15	01	66	66	66	66	8E	00	01	00	22	AF	D6	BF	DC	5E	5D
16 – 31	2D	B3	B0	EE	94	AA	1B	BF	74	E7	31	DE	53	E4	0C	51
32 – 47	BF	76	98	BC	D8	8D	5F	AF	94	C5	91	7D	E8	FA	3C	AA
48 – 63	D3	4C	69	AD	56	62	1D	E7	6A	FD	3A	E4	47	74	1A	73
64 – 79	FB	4A	2E	6B	49	E1	CB	37	89	97	C2	DC	2A	72	F4	71
80 – 95	AE	B4	5E	A6	98	73	2F	3A	1D	B7	12	9A	E0	A5	B9	48
96 – 111	47	B5	5F	8E	10	84	90	F3	66	50	55	87	9C	8F	D4	45
112 – 127	81	C6	D1	78	9B	72	40	66	32	D9	DB	7A	13	10	B1	DA
128 – 143	92	83	92	8F	57	9E	85	B1	4E	37	28	46	00	00	-	-

Tabelle 50: Balisentelegramm senden (2 Tlg. – Test Latenzzeit)

Byte Nr.	Daten															
0 – 15	02	00	00	00	00	20	00	02	01	00	E8	03	00	00	E8	03
16 – 31	00	00	01	00	E8	03	00	00	E8	03	00	00	D2	03	00	00

Tabelle 51: Balisentelegramm senden (3 Tlg. – Test Latenzzeit)

Byte Nr.	Daten															
0 – 15	02	00	00	00	00	2A	00	03	01	00	E8	03	00	00	E8	03
16 – 31	00	00	01	00	E8	03	00	00	E8	03	00	00	01	00	E8	03
32 – 47	00	00	E8	03	00	00	B4	05	00	00	-	-	-	-	-	-

Tabelle 52: Balisentelegramm senden (4 Tlg. – Test Latenzzeit)

Byte Nr.	Daten															
0 – 15	02	00	00	00	00	34	00	04	01	00	E8	03	00	00	E8	03
16 – 31	00	00	01	00	E8	03	00	00	E8	03	00	00	01	00	E8	03
32 – 47	00	00	E8	03	00	00	01	00	E8	03	00	00	E8	03	00	00
48 – 63	96	07	00	00	-	-	-	-	-	-	-	-	-	-	-	-

Tabelle 53: Latenzzeiten des DSP - Systems

Tlg. Format	TX-Fehler	Items pro Sequenz	Latenzzeit in µs	Differenz in µs
Short	-	1	360	360
Short	-	2	700	340
Short	-	3	1040	340
Short	-	4	1360	320
Long	-	1	936	936
Long	-	2	1856	920
Long	-	3	2780	924
Long	-	4	3700	920

Aus der Tabelle 53 ist zu erkennen, dass die Latenzzeit des Systems mit der Anzahl der Elemente pro Sequenz linear ansteigt. Zusätzlich ist diese Differenz unterschiedlich groß für lange und kurze Telegramme. Wird nun zusätzlich zur reinen Latenzzeit T_{LatDSP} des DSP - Systems die Latenzzeit der RS 232 Übertragung $T_{LatRS232}$ berücksichtig, so kann eine Formel entwickelt werden, mit der die erste Zeit der Sequenzdefinition, also T_{Pause} des ersten Eintrags der Sequenz um alle Latenzzeiten verkürzt werden kann.

$$T_{PauseKorr} = T_{Pause} - (N_{short} \cdot T_{LatShort} + N_{long} \cdot T_{LatLong}) - \frac{N_{PaketLength} \cdot 10}{R_{Rs232RailSiTe}} \qquad \textbf{(20)}$$

Formel 20: Korrektur der Latenzzeit des DSP Systems

Dabei beschreiben N_{short} und N_{long} jeweils die Anzahl langer oder kurzer Telegramme in der Sequenz. $T_{LatShort}$ und $T_{LatLong}$ haben jeweils den Wert der Latenzzeit, die durch einen Sequenzeintrag mit langen, oder kurzem Telegramm erzeugt wird. $N_{PacketLength}$ ist schließlich die Anzahl der Bytes des RS 232 Pakets und $R_{Rs232RailSiTe}$ die Datenrate, mit der die Pakete vom RailSiTe an das DSP - System übertragen werden.

Der Wegfehler, der durch diese Latenzzeit erzeugt wird, kann durch die maximale Geschwindigkeit berechnet werden, die das Labor als simulierte Geschwindigkeit unterstützen soll. Diese Geschwindigkeit beträgt v_{Max} = 500 km/h. Über die folgende Formel können die Fehler für die gemessenen Geschwindigkeiten berechnet werden

$$F_{X\max} = T_{LatDSP} \cdot v_{Max} \qquad \textbf{(21)}$$

Formel 21: Berechnung des Wegfehlers durch DSP Latenzzeit

Für die gemessenen Zeiten ergeben sich dann folgende Wegfehler:

Tabelle 54: maximale Wegfehler für DSP Latenzzeiten

Tlg. Format	TX-Fehler	Items pro Sequenz	Latenzzeit in µs	Maximaler Wegfehler in m
Short	-	1	360	0,050
Short	-	2	700	0,097
Short	-	3	1040	0,144
Short	-	4	1360	0,189
Long	-	1	936	0,130
Long	-	2	1856	0,258
Long	-	3	2780	0,386
Long	-	4	3700	0,514

Aus der Tabelle ist zu ersehen, dass ein maximaler Wegfehler bei langen Telegrammen und vier Balisen innerhalb einer Sequenz von F_{Xmax} = 0,514 m auftritt. Da dieser Fehler nur eine Verschiebung der Balisengruppe bedeutet, aber die Balisengruppe in sich konsistent bleibt, kann dieser Fehler nur bei Balisengruppen kritisch werden, auf die über eine zweite Balisengurppe mittels eines „Link“ Befehls verwiesen wird.

Mittels des „Link“ Befehls kann eine Strecke definiert werden, nach der eine Balisengruppe empfangen werden muss, um gültig zu sein. Zusätzlich wird für diese Strecke ein Toleranzfenster definiert. Dieses gibt an in welchem Fenster um die angegebene Position der gelinkten Balise diese empfangen werden darf. Dieses Toleranzfenster ist laut [12] (Seite 54) mit einer räumlichen Auflösung von 1 m definiert.

Da laut [11] (Seite 95) die maximale Anzahl von Balisen innerhalb einer Balisengruppe auf acht Eurobalisen festgesetzt ist, ergibt sich ein maximaler Wegfehler für eine solche Gruppe von:

$$F_{X\max} = 0{,}130m \cdot 8 = 1{,}040m \qquad \textbf{(22)}$$

Formel 22: maximaler Wegfehler bei acht Balisen in einer Gruppe

Auf Grund des minimalen Toleranzfenster für gelinkte Balisen von 1 m [12], kann die Latenzzeit des DSP - Systems, die in diesem Fall nur zu einem Wegfehler von 1,040 m führt, vernachlässigt werden, da diese räumliche Differenz von einem realem Fahrzeuggerät nicht erkannt werden kann.

Sollten zu einem späteren Zeitpunkt durch die Einbringung besonderer Fehlermodelle oder Softwareerweiterungen größere Latenzzeiten auftreten, die diese Bedingung verletzen, so muss zwangsweise eine Korrektur durchgeführt werden. Diese darf dann nicht pauschal für alle Fehlermodelle die gleichen Parameter verwenden, sondern muss vielmehr für jedes Fehlermodell eine individuelle Korrektur durchführen.

Eine Vermeidung der Latenzzeit durch sofortiges Starten des Timer-1 bei Empfang des Pakets **RS2DSP_DefAndRunSeq** ist auf Grund der Mehrkanaligkeit des DSP - Systems nicht möglich.

11.3 Schaltplan FSK Modulator (neu)

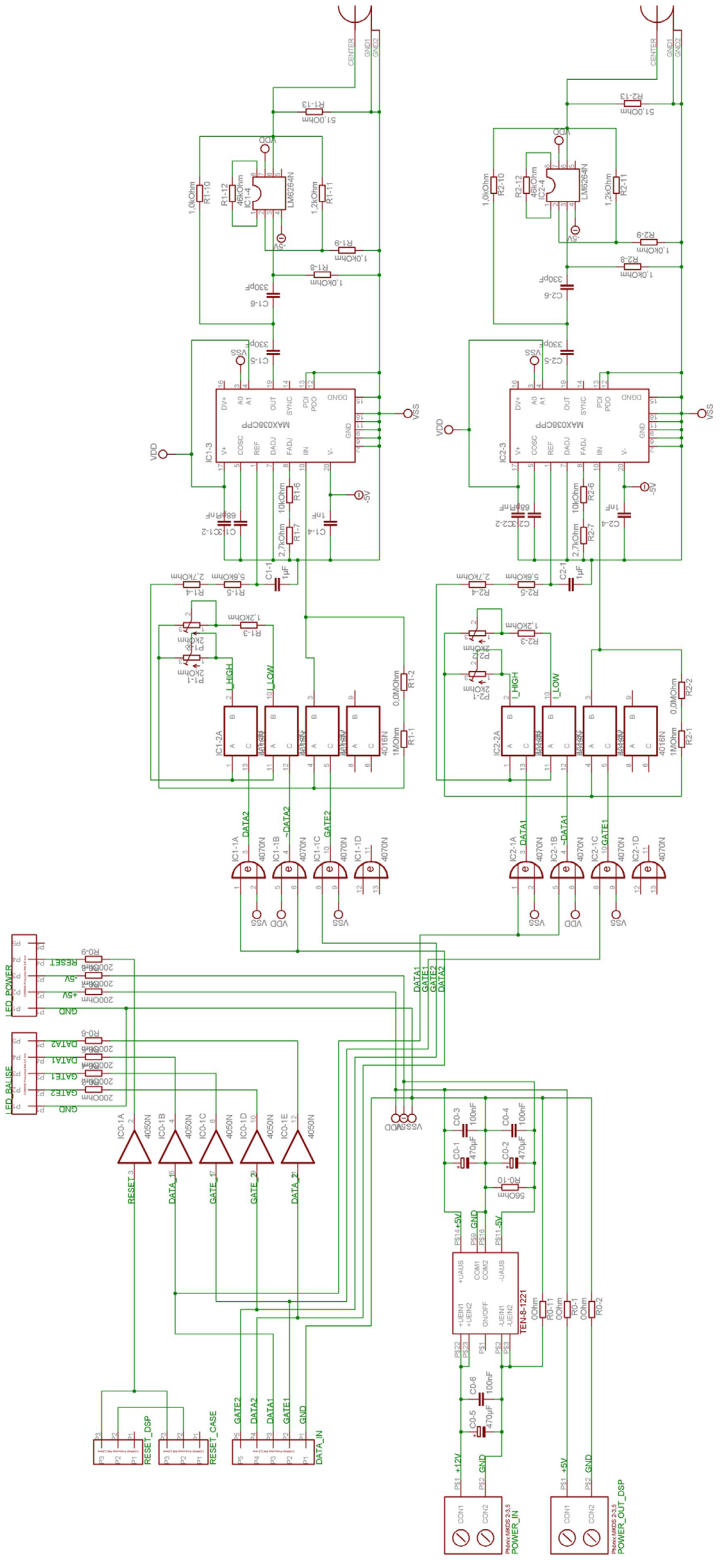

Abbildung 56: Schaltplan FSK Modulator (neu)

11.4 Layout FSK Modulator (neu)

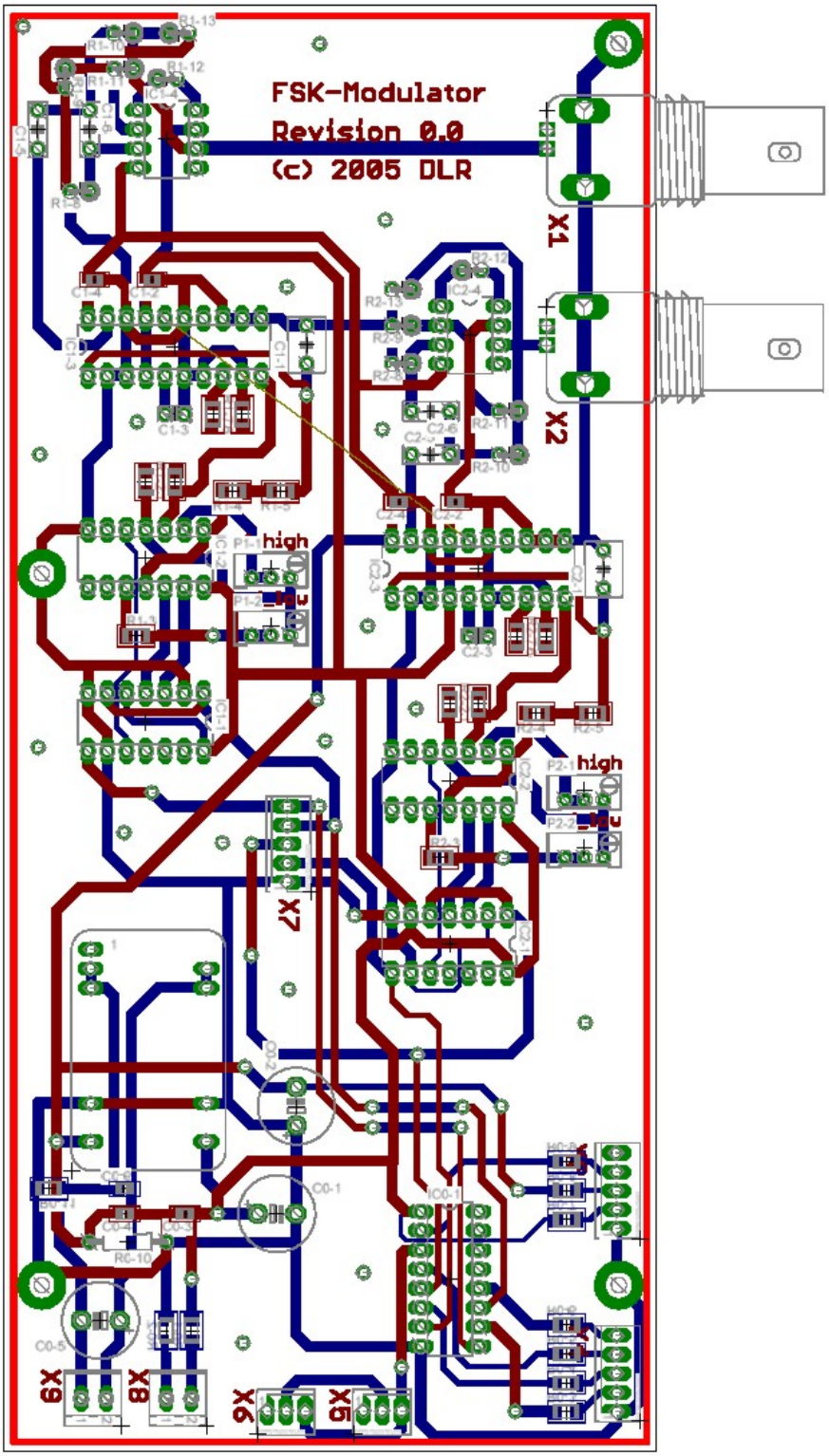

Abbildung 57: Layout FSK Modulator (neu)

11.5 Oszilloskopbilder der TX Fehlermodelle

11.5.1 Einzelbitfehler

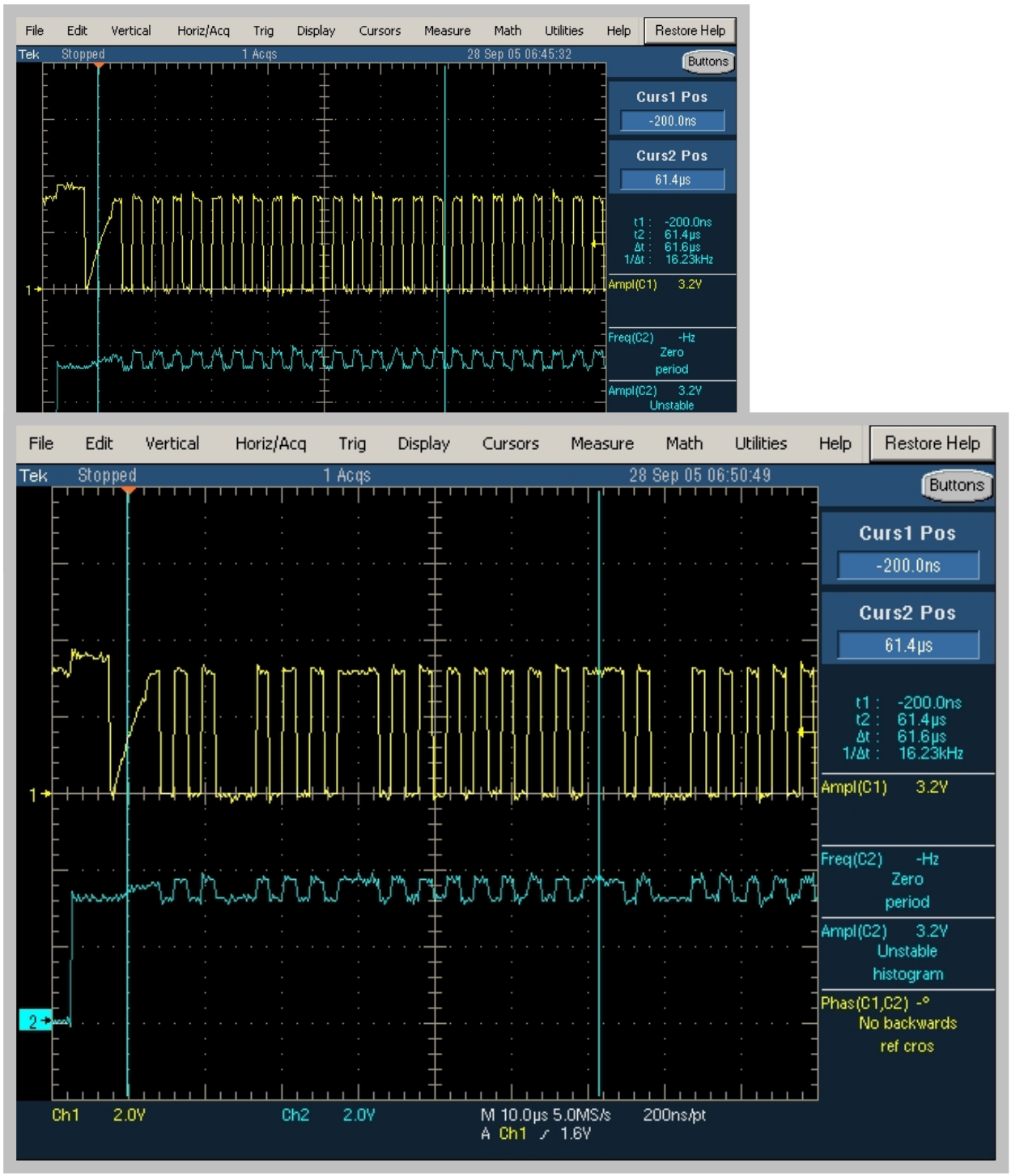

Abbildung 58: Einzelbitfehler (Oszilloskopbild)

11.5.2 Bitburstfehler

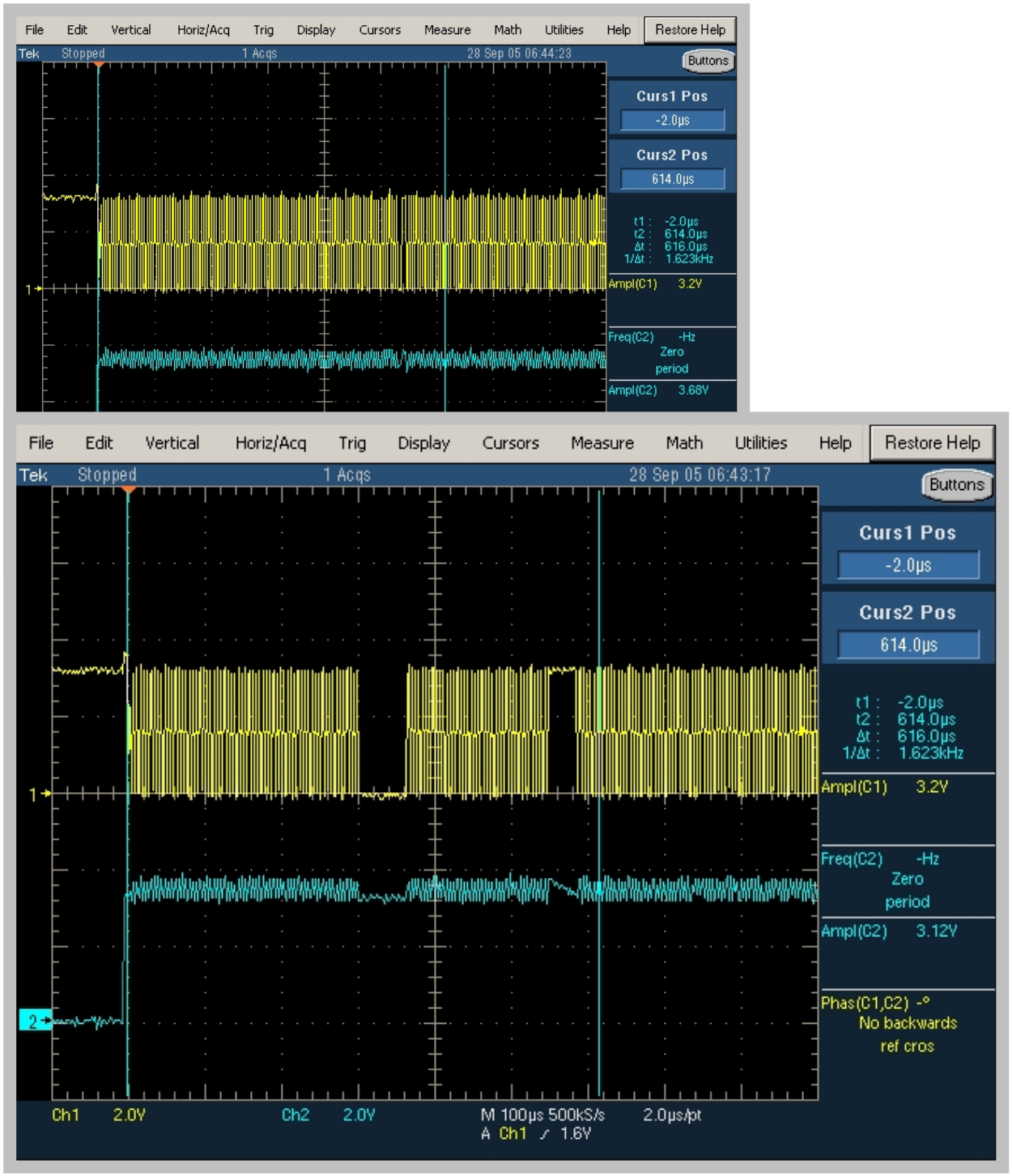

Abbildung 59: Bitburstfehler (Oszilloskopbild)

11.5.3 Bitslip und Bitinsertion

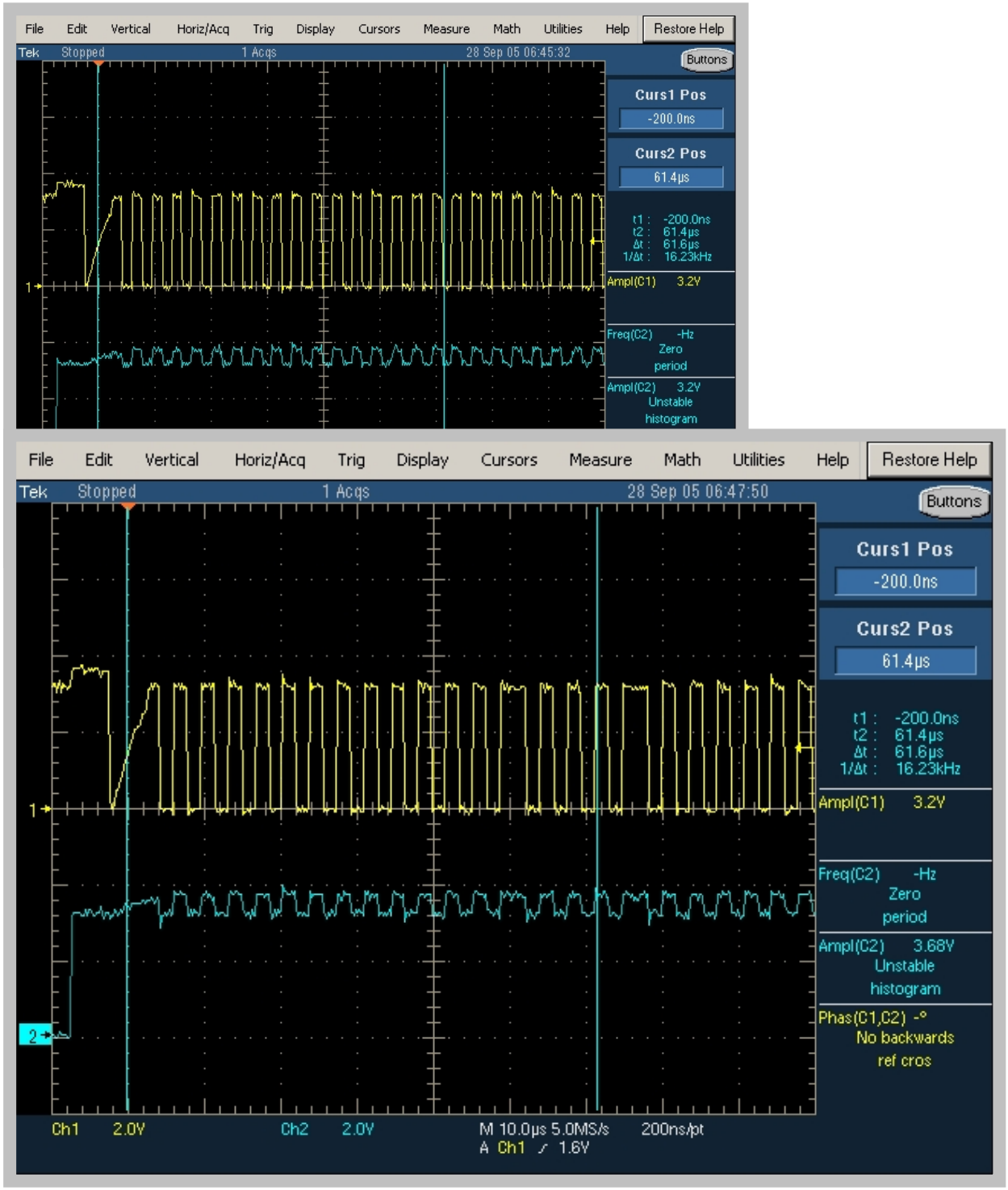

Abbildung 60: Bitslip Bitinsertion (Oszilloskopbild)

11.5.4 Oversampling

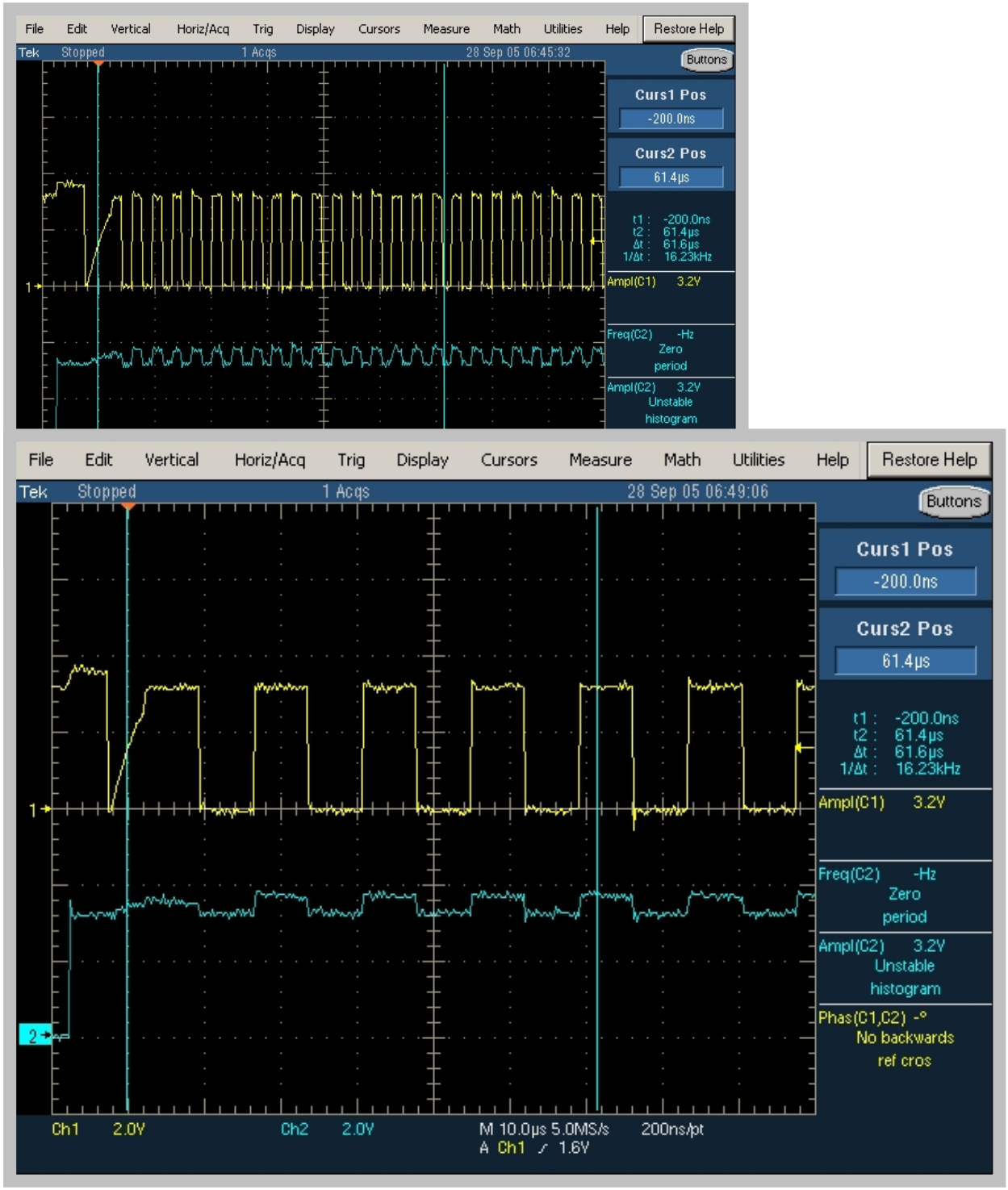

Abbildung 61: x-fach Oversampling (Oszilloskopbild)

11.5.5 Undersampling

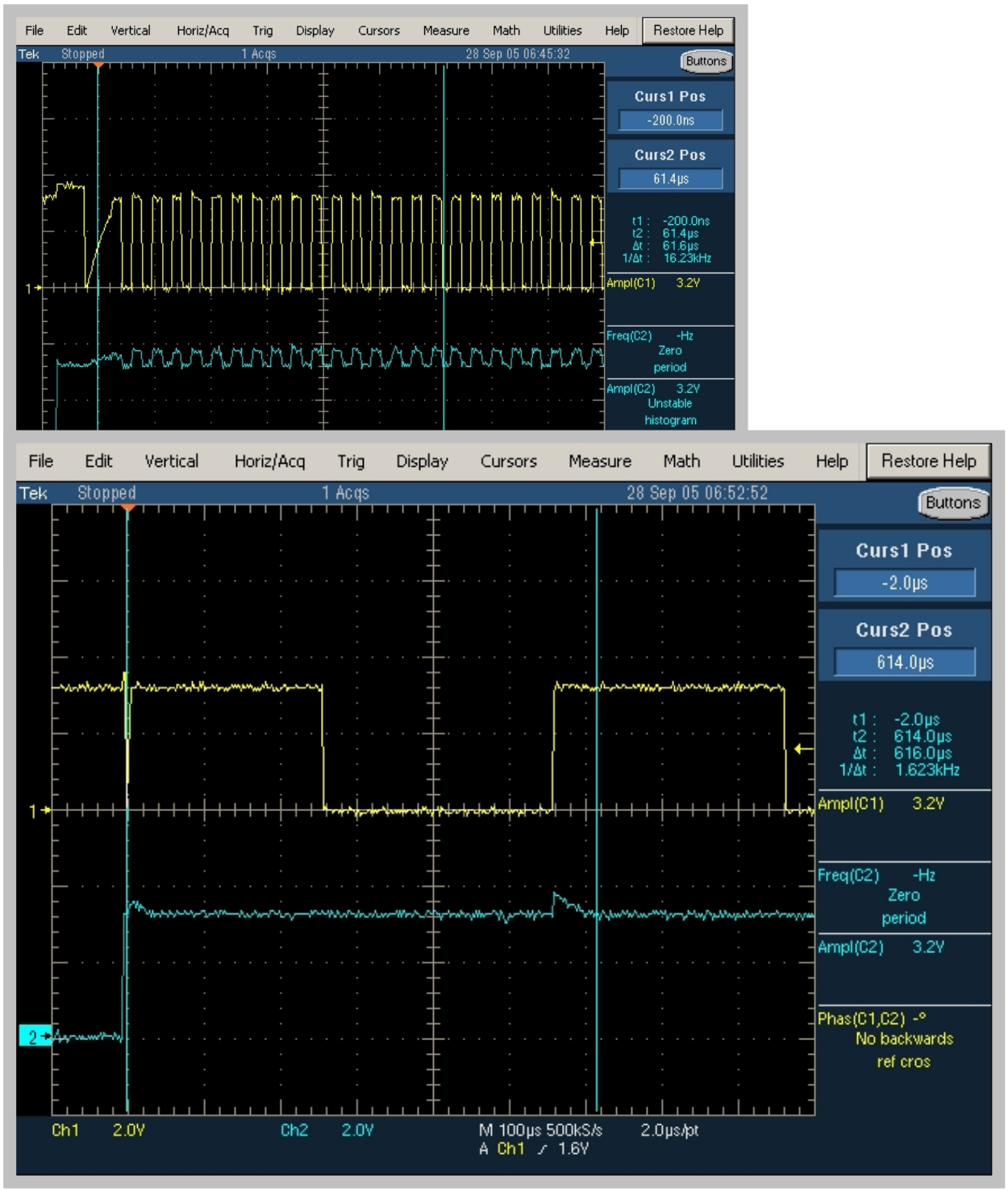

Abbildung 62: x-fach Undersampling (Oszilloskopbild)