RS232 Treiber

Für das studentische Projekt *Sichere Eisenbahnsteuerung*

Datum	06.05.2010
Quelle	Google Code → Dokumente → 02_Design → 02.02_Moduldesign
Autoren	Jan-Christopher Icken
Version	2.0
Status	In Bearbeitung

1 Historie

Version	Datum	Autor	Bemerkung
0.10	12.12.09	Oliver Gatti	Grundsätzliche Informationen über XpressNet und das LI101F-Interface erläutert.
0.11	14.12.09	Oliver Gatti	Diverse kleine Änderungen.
0.2	15.12.09	Oliver Gatti	Empfang von Daten hinzugefügt und bestehende Fragen zur Hardware geklärt.
0.3	16.12.09	Oliver Gatti	Senden von Daten hinzugefügt
0.4	21.12.09	Oliver Gatti	Unklarheiten geklärt. Noch offen: Adressen der Weichen etc.
0.5	22.12.09	Oliver Gatti	Adressen für Schaltdecoder hinzugefügt (6.1.3)
1.0	02.02.10	Oliver Gatti	Review eingearbeitet
1.1	02.02.10	Oliver Gatti	Stoppen einer Lok entfernt. Initialisierung der Seriellen Schnittstelle wird nun mit Keil uVision durchgeführt. Schalten von Weichen geändert (Sie müssen zunächst deaktiviert werden, bevor sie geschaltet werden können). char durch byte ersetzt. 3 Lokale Variablen hinzugefügt.
1.2	04.02.10	Oliver Gatti	Shared-Memory aktualisiert. Überall im Dokument die Array Beschreibung des Shared Memory in die aktuelle Struct Version umgeschrieben. Kapitel „Probleme“ hinzugefügt. Variablennamen in den Aktivitätsdiagrammen angepasst.
1.3	05.05.10	Jan-Christopher Icken	Abgleich mit dem vorhandenen Quellcode, Entfernung der äußeren Schnittstellen, Anpassung des Layouts und Korrektur von Rechtschreibfehlern, Kapitel 7 vervollständigt
2.0	24.06.10	Jan-Christopher Icken	Dokument freigegeben

2 Inhaltsverzeichnis

3 Einleitung

Dieses Modul dient als Schnittstelle zwischen der Software und dem Gleissystem.Der Microcontroller ist über die RS232 Schnittstelle mit dem LI101F Interface verbunden. Dieses Modul kommuniziert also direkt mit dem LI101F und damit indirekt mit der Zentrale, die das DCC-Signal erzeugt.

Das verwendete Datenprotokoll ist XpressNet Version 3.0.

Der RS232-Treiber liest aus dem Shared-Memory den abzuschickenden Streckenbefehl aus und konvertiert ihn in das entsprechende XpressNet-Format.

4 Referenzierte Dokumente

Grundlage für dieses Moduldesign bilden folgende Dokumente:

- Aulis >
  Projektverzeichnis >
  Schnittstellendokumentation >
  XpressNet LI101F Beschreibung
  > **xpressnet_li101f.pdf**
  > **c515c Datasheet.pdf**
  > **c515c User Manual.pdf**
- Aulis >
  Projektverzeichnis >
  Implementierung >
  C für den Mikrocontroller 90C515C
  > **C.pdf**

5 Architektur

5.1 Die RS232-Schnittstelle

5.1.1 Baudrate

XpressNet weist grundsätzlich eine Baudrate von 62.5 Kbit/s auf. Das LI101F Interface kapselt diese Datenrate von der RS232-Schnittstelle ab und gibt die Möglichkeit zwischen vier Baudraten zu wählen. Die Konfiguration geschieht über den PC per mitgelieferter Software(„Lenz LI101F Tool“).

- 19200
- 38400
- 57600
- 115200

Ich lege die Baudrate ad-hoc auf 19200 bzw 19.2 Kbit/s fest.

5.1.2 USART Mode

Der USART-Mode gibt an wie die einzelnen Datenpakete übertragen werden.

In diesem Fall wird der Mode 1 benutzt, da die Kommuikation auf folgendem Format basiert:

10 Bits insgesamt, 8 Datenbits, 1 Startbit(0) , 1 Stoppbit(1)

5.1.3 Senden

Das zu sendende Byte wird in das Senderegister SBUF geschrieben, wodurch implizit die Datenübertragung initiiert wird. Das heißt, nachdem die Daten in den Buffer geschrieben worden sind, werden sie automatisch übertragen. Nachdem das Byte erfolgreich übertragen worden ist, wird das Register TI auf 1 gesetzt.

5.1.4 Empfangen

Das Empfangen der Daten ist auf Hardwarebasis gelöst. Nachdem ein Byte empfangen worden ist, wird das Register RI auf 1 gesetzt. Das Datenbyte befindet sich im Buffer SBUF.

Folgende Bedingungen müssen erfüllt sein, damit ein Datenbyte empfangen werden kann:

1. Register RI = 0
2. Register SM2 = 0 oder das empfangene Stoppbit ist 1

5.1.5 Hardware-Handshake

Das LI101f erlaubt das Senden vom Microcontroller nur, wenn das CTS Signal gesetzt ist.

Der Microcontroller unterstützt diesen Anschluss nicht direkt. Die Leitung muss separat an einen Pin geführt werden. Der Zugriff auf diesen Pin findet über die Betriebsmittelverwaltung, die die Konstante RS232TREIBER_CTSPIN bereitstellt, statt.

5.2 Datenkommunikation

5.2.1 Antworten des LI101F

Das Interface liefert unabhängig von der Zentrale bzw. dem XpressNet-Format folgende Information:

Headerbyte	Meldung	X-Or	Bedeutung
0x01	0x01	0x00	Fehler zwischen Interface und PC (ca. 250ms Timeout bei Datenübertragung PC an LI)
0x01	0x02	0x03	Fehler zwischen Interface und Zentrale (ca. 250ms Timeout bei Übertragung LI an Zentrale)
0x01	0x03	0x02	Unbekannter Fehler (Zentrale adressierte LI101F mit Quittierungsaufforderung)
0x01	0x04	0x05	Befehl ist an Zentrale geschickt oder Zentrale adressiert LI101F wieder
0x01	0x05	0x04	Zentrale adressiert LI101F nicht mehr
0x01	0x06	70x0	Puffer-Überlauf im LI101F

Bis auf (0x01 0x04 0x05) (1/4/5) stehen alle Informationen für entstandene Fehler, die im Shared-Memory als Fehler quitiert werden und zu einem NOT-Aus führen.

(1/4/5) sagt aus ob ein Streckbefehl erfolgreich an die Zentrale gesendet worden ist. Diese Informationen sollte empfangen worden sein, bevor ein weiterer Befehl abgeschickt wird. Falls das nicht der Fall ist, wird ein Fehler in den Shared-Memory geschrieben, der zu einem Not-Aus führt.

[Für Informationen über die erzeugten Fehler siehe 5.3.1]

5.2.2 XpressNet-Format

Die empfangenen XpressNet-Daten beginnen mit einem Headerbyte. Der obere Nibble kennzeichnet um welchen Befehl es sich handelt. Der untere Nibble gibt Aufschluss darüber wieviele Bytes nach dem Header-Byte folgen. Hierbei ist zu beachten, dass der Wert um einen erhöht betrachtet werden muss, da er nicht das X-Or-Byte beinhaltet.

Als letztes Byte weist jeder XpressNet-Befehl ein X-Or Byte auf. Dieses wird ermittelt durch eine X-Or Verküpfung aller anderen Bytes.

5.2.3 Fahrbefehle (128 Stufen)

Es wird unterschieden zwischen vier verschiedenen Fahrstufen (14, 27, 28 und 128).

Der RS232-Treiber basiert auf der Annahme, dass mit 128 Stufen gearbeitet wird. Das Format für den Fahrbefehl sieht dann wie folgt aus:

	Header	**Kennung**	**Daten 1**	**Daten 2**	**Daten 3**	**X-Or**
Binär	1110 0100	0001 0011	Address High	Address Low	RVVV VVVV	X-Or
Hex	0xE4	0x13	AH	AL	RV	X-Or

IIn diesem Projekt arbeiten wir nur mit 2 Loks und nutzen entsprechend geringe Adressbereiche. Dadurch ergibt sich, dass
AH = 0x00
und
AL werte zwischen 0x00 und 0x63 aufweist.

(Für Adressen < 100)

Das Daten 3 Byte gibt nun die eigentliche Information über die zu fahrende Geschwindigkeit:
Daten 3:
Fahrstufe 0: 0 0 0 0 0 0 0
Nothalt: 0 0 0 0 0 0 1
Fahrstufe 1: 0 0 0 0 0 1 0
..
Fahrstufe 127: 1 1 1 1 1 1 1

Das MSB entscheidet darüber ob die Lok vorwärts oder rückwärts fahren soll:

R = 1 = vorwärts
R = 0 = rückwärts

5.2.4 Schaltbefehle

	Header	**Daten 1**	**Daten 2**	**X-Or**
Binär	0101 0010	AAAA AAAA	1000 (D1)BB(D2)	X-Or
Hex	0x52	Adresse	0x80 + (D1)BB(D2)	X-Or

Adresse einer Weiche (10 Bit): AAAA AAAA BB

D2 = 0 =>Ausgang 1 der Weiche gewählt.

D2 = 1 =>Ausgang 2 der Weiche gewählt.

D1 = 0 => Ausgang deaktivieren.

D1 = 1 => Ausgang aktivieren.

5.2.4.1 Weiche schalten

Um eine Weiche in eine andere Position zu bewegen muss zunächst der andere Ausgang deaktiviert werden. Erst im Anschluss kann der gewünschte Ausgang aktiviert und somit geschaltet werden.

Gerade aus

D2 = 0 und D1 = 0

D2 = 1 und D1 = 1

Abbiegen

D2 = 1 und D1 = 0

D2 = 0 und D1 = 1

5.2.4.2 Entkupplungsdekoder schalten

Hoch

Erst muss der Ausgang deaktiviert werden:

D2 = 0 und D1 = 0

Anschließend muss der Befehl zum aktivieren abgeschickt werden, der dann den Entkuppler hebt:

D2 = 0 und D1 = 1

Runter

Automatisch

5.2.5 Sonstige

Alle sonstigen eventuell empfangenen Informationen spielen keine Rolle und müssen im Rahmen dieses Moduls nicht weiter beachtet werden.

6 Dynamisches Verhalten

6.1 Globale Variablen

6.1.1 Betriebsmittelverwaltung

const byte RS232TREIBER_CTSPIN

Auszulesende Werte: 1 (Senden erlaubt), 0 (Senden verboten)

6.1.2 Shared-Memory

Streckenbefehl EV_RS232_streckenbefehl

Der Struct Streckenbefehl ist in Betriebsmittelverwaltung.h definiert und enthält folgende Variablen:

- byte Lok
- byte Weiche
- byte Entkoppler
- byte Fehler

6.1.3 Allgemein

(Die Konstanten für die Geschwindigkeit V_* müssen noch experimentell bestimmt werden)

byte	V_Abkuppeln	=	0x08
byte	V_Ankuppeln	=	0x0F
byte	V_Fahrt	=	0x7F
byte	Lok1_address	=	0x01
byte	Lok2_address	=	0x02
byte	W1_address	=	0x03
byte	W2_address	=	0x04
byte	W3_address	=	0x02
byte	EK1_address	=	0x06
byte	EK2_address	=	0x05

6.1.4 Modulspezifisch

Datentyp	Name der Variable	Beschreibung
byte[3]	receiveBuffer	In diesem Array werden die empfangenen Datenbytes abgelegt.
byte	waitForSentAnswer	Nachdem ein Streckenbefehl an das LI101F gesendet worden ist, muss auf eine Quittierungsantwort gewartet werden bevor ein neuer Befehl abgesetzt werden kann (siehe 5.2.1). Wenn die Variable 0x01 ist, dann wird noch gewartet.
byte	calledWithoutSentAnswer	Nachdem ein Streckenbefehl an das Interface gesendet worden ist, muss gewartet werden bis der Befehl quitiert wird. Diese Variable zählt wie oft der RS232-Treiber aufgerufen worden ist, bevor die Antwort empfangen wurde.
byte	bytesToReceive	Ein Counter, der anzeigt wieviele (nicht Header) Bytes noch empfangen werden müssen.
byte	li101HeaderReceived	Dieses Modul muss nur Antworten des LI101f empfangen. Diese sind immer 3 Byte lang. Alle restlichen Nachrichten, die ggf. empfangen werden (Broadcasts der Maus etc.) sind unwichtig und nicht weiter zu beachten. Dennoch muss das Empfangsflag RI auf 0 gesetzt werden, um alle Bytes empfangen zu können. Ist diese Variable gesetzt (0x01) werden die empfangenen Bytes nicht verworfen sondern in receiveBuffer abgelegt.
byte	readyToSend	Nachdem ein Streckenbefehl abgesendet worden ist, muss auf eine Quittierung vom LI101F gewartet werden. So lange gewartet wird, hat die Variable den Wert 0x00. Wenn die Quitterung empfangen worden ist, wird die Variable auf 0x01 gesetzt. Damit kann ein neuer Streckenbefehl abgesetzt werden.
byte	ctsPinFalse	Diese Variable dient zur Flusskontrolle. Wenn das LI101F signalisiert nicht bereit zu sein, wird diese Variable inkrementiert. Nach drei Versuchen wird ein Fehler in den Shared-Memory geschrieben.
byte	sendeCounter	Signalisiert wieviele Bytes noch abgeschickt werden

		müssen. Dient außerdem als Pointer, welches Datenbyte abgeschickt werden muss (siehe sendeBuffer).
byte[6]	sendeBuffer	Beinhaltet den jeweilig abzusenden Streckenbefehl, wobei das LSB (Das zuerst zu sendende Byte) an der Position (Anzahl der Bytes – 1) steht. z.B.: Es müssen 3 Bytes verschickt werden. Das erste abzuschickende Byte liegt in sendeBuffer[2]. Das zuletzte abzuschickende in sendeBuffer[0].
byte const	MaxFailures	Wert = 0x05. Gibt an wie oft Verzögerungen beim Senden auftreten dürfen bevor ein Fehler in den Shared-Memory geschrieben wird.
byte	rs232Enabled	0x01 falls Master Microcontroller. 0x00 falls Slave Microcontroller.
byte	entkupplerActive	Wird dazu genutzt zu unterscheiden, ob der zu schaltende Dekoder schon aktiviert werden kann oder zunächst noch deaktiviert werden muss.

6.2 Interne Funktionen

Name:	konvertEntkoppler()
Vorbedingung:	Streckenbefehl im Shared Memory
Parameter:	-
Rückgabe:	-
Nachbedingung:	Streckenbefehl aus dem Shared-Memory ins XpressNet-Format konvertiert
Fehlerfall:	

Name:	konvertLok()
Vorbedingung:	Streckenbefehl im Shared Memory
Parameter:	-
Rückgabe:	-
Nachbedingung:	Streckenbefehl aus dem Shared-Memory ins XpressNet-Format konvertiert
Fehlerfall:	

Name:	konvertWeiche()
Vorbedingung:	Streckenbefehl im Shared Memory
Parameter:	-
Rückgabe:	-
Nachbedingung:	Streckenbefehl aus dem Shared-Memory ins XpressNet-Format konvertiert
Fehlerfall:	

6.3 Erzeugte Fehler

In den Shared-Memory, in die Variable RS232_EV_streckenbefehl.Fehler, können folgende Fehler Inhalte geschrieben werden:

Binär	Hex	Beschreibung
0000 0000	0x00	Kein Fehler
0000 0001	0x01	Fehler zwischen Interface und PC
0000 0010	0x02	Fehler zwischen Interface und Zentrale
0000 0011	0x03	unbekannter Fehler (Zentrale adressierte LI101F mit Quittierungsaufforderung)
0000 0100	0x04	Keine Antwort mehr vom Interface – Daten wurden nicht verschickt
0000 0101	0x05	Zentrale adressiert LI101F nicht mehr
0000 0110	0x06	Puffer-Überlauf im LI101F

6.4 Initialisierung der RS232-Schnittstelle

In diesem Abschnitt wird gezeigt, wie die die Serielle Schnittstelle (RS232) des Microcontrollers auf die Anforderungen für dieses Modul initialisiert wird.

ACHTUNG: Der aktuelle Stand nutzt die Initialisierung der Schnittstelle (die Baudrate und die erforderlichen Register) von Keil uVision. Interrupts müssen nach wie vor in der Initialisierungsroutine aktiviert werden.

6.4.1 Setzen des Betriebsmodus

Um den Betriebsmodus der Schnittstelle auszuwählen müssen zwei BITS des SCON (Special Function Register) entsprechend gesetzt werden:

SM0 (SCON.7): 0

SM1 (SCON.6): 1

Zusätzlich werden SM2, TI und RI auf 0, sowie REN auf 1 gesetzt.

6.4.2 Setzen und Erzeugen der Taktrate

Die für die Datenübertragung genutzte Taktrate wird vom internen Baudraten-Generator erzeugt.

SMOD (PCON.7) = 1

SREL muss mit einem geeigneten Wert initialisiert werden um der Baudrate von 19200 entsprechen zu können:

$$baudrate = \frac{(2^{smod} * Oscillator\ frequenzy)}{(32 * baud\ rate\ generator\ overflow\ rate)}$$

$$baud\ rate\ generator\ overflow\ rate = 2^{10} - SREL$$

$$SREL = SRELH.1-0, SRELL.7-0$$

$$baud\ rate\ generator\ overflow\ rate = \frac{(2^1 * 10000000)}{(32 * baudrate)} = 32.552 = 33$$

$$SREL = 2^{10} - 33 = 1024 - 33 = 991 = 0x3DF = 1111011111$$

SRELH.1 = 1

SRELH.0 = 1

SRELL.7-0 = 1101 1111

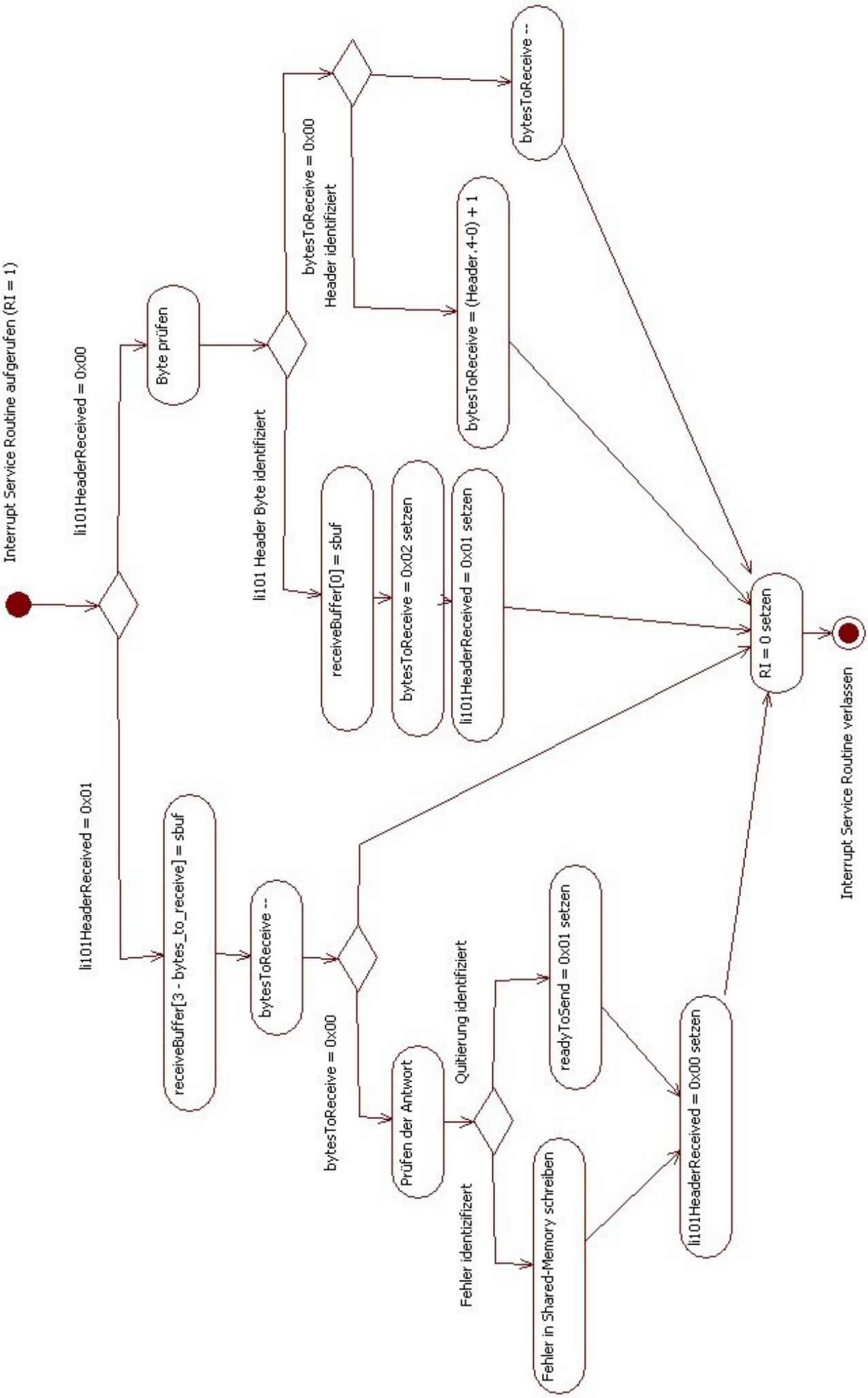
Um den Generator zu aktivieren muss das Bit BD (ADCON0.7) gesetzt werden.

6.4.3 Interrupts

Um die Interrupts der Seriellen Schnittstelle zu aktivieren muss ES = 1 (IEN0.4) gesetzt werden.

6.5 Empfangen von Daten

Der Empfang von Daten geschieht asynchron. Auf den Eingang von Daten (RI = 1) muss mittels Aufruf einer Interrupt-Service-Routine reagiert werden um das empfangene Datenbyte zu sichern, bevor ein neues Empfangen wird. Folgendes Aktivitätsdiagramm zeigt den Ablauf:


Interrupt Service Routine aufgerufen (RI = 1)
li101HeaderReceived = 0x00
Byte prüfen
li101 Header Byte identifiziert
bytesToReceive = 0x00
Header identifiziert
bytesToReceive --
bytesToReceive = (Header.4-0) + 1
receiveBuffer[0] = sbuf
bytesToReceive = 0x02 setzen
li101HeaderReceived = 0x01 setzen
li101HeaderReceived = 0x01
receiveBuffer[3 - bytes_to_receive] = sbuf
bytesToReceive --
bytesToReceive = 0x00
Prüfen der Antwort
Quitierung identifiziert
readyToSend = 0x01 setzen
Fehler identizifizert
Fehler in Shared-Memory schreiben
li101HeaderReceived = 0x00 setzen
RI = 0 setzen
Interrupt Service Routine verlassen

6.6 Senden von Daten

In diesem Kapitel wird der gesamte Ablauf vom Aufruf der RS232-Treibers bis zur Rückgabe der Kontrolle an die Betriebsmittelverwaltung beschrieben. Am Ende des Kapitels 6.5 findet sich ein beschreibendes Aktivitätsdiagramm.

6.6.1 EV_RS232_streckenbefehl prüfen

Das Shared Memory enthält den struct EV_RS232_streckenbefehl. Dabei beinhaltet

- EV_RS232_streckenbefehl.Lok : Lok Befehle
- EV_RS232_streckenbefehl.Weiche : Weichenbefehle
- EV_RS232_streckenbefehl.Entkoppler Kuppelbefehle

Die Ergebnisvalidierung füllt diese Shared-Memory-Variable erst, wenn alle drei Einträge den Wert 0xFF aufweisen. Der RS232-Treiber setzt den entsprechenden Eintrag auf diesen Wert, nachdem der jeweilige komplett Streckenbefehl verschickt worden ist.

Die EV muss stets alle drei Elemente des Arrays füllen. Der Wert 0xFF wird dabei verwendet, um zu signalisieren, dass keine Aktion durchgeführt werden muss.

Wird der RS232-Treiber aufgerufen, wird entweder der Reihenfolge nach 0, 1, 2 im Shared-Memory geprüft ob ein Streckbefehl abgesetzt werden soll oder ob noch Bytes abzuschicken sind (sende_counter ungleich 0x00).

Nachdem ein Befehl an das LI101F geschickt worden ist, wird der entsprechende Eintrag im Shared-Memory auf 0xFF gesetzt.

6.6.2 Konvertierung der Streckbefehle

In den folgenden drei Kapiteln wird erläutert, wie die Streckbefehle des Systems in das XpressNet-Format konvertiert werden.

6.6.2.1 Lok steuern

EV_RS232_streckenbefehl.Lok.0 = Wahl des Zugs:

- xxxx xxx0 → Lok1_address
- xxxx xxx1 → Lok2_address

EV_RS232_streckenbefehl.Lok.1 = Wahl der Richtung

- xxxx xx0x → Rückwärts(0)
- xxxx xx1x → Vorwärts(1)

EV_RS232_streckenbefehl.Lok.7-2 = Geschwindigkeitstufe, wobei nur 0, 1 ,2 , 3 in Benutzung sind. Das führt zu folgendem Verhalten:

- 0000 00xx → Stop der Lok
- 0000 01xx → V_Abkuppeln
- 0000 10xx → V_Ankuppeln
- 0000 11xx → V_Fahrt

Das führt zur Generierung folgendes XpressNet-Befehls, wenn die Lok nicht gestoppt wird:

	Header	**Kennung**	**Daten 1**	**Daten 2**	**Daten 3**	**X-Or**
Binär	1110 0100	0001 0011	0000 0000	Lokadresse	DirectionSpeed	X-Or
Hex	0xE4	0x13	0x00	Lokadresse	DirectionSpeed	X-Or

Lokadresse = Lok1_address oder Lok2_address

DirectionSpeed = ((0x02 **AND** EV_RS232_streckenbefeh.Lokl) << 6)
OR V_Abkuppel/Ankuppeln/Fahrt

EV_RS232_streckenbefehl..Weiche.0 = Weichenposition

- xxxx xxx0 → Geradeaus
- xxxx xxx1 → Abbiegen

EV_RS232_streckenbefehl.Weiche.7-1 = Weichenadresse

Das führt zur Generierung folgendes XpressNet-Befehls:

	Header	**Daten 1**	**Daten 2**	**X-Or**
Binär	0101 0010	AAAA AAAA	1000 (D1)BB(D2)	X-Or
Hex	0x52	Adresse	0x80 + (D1)BB(D2)	X-Or

Weichengruppe = (W1/2_address >> 2)

Für D1 und BB siehe Kapitel 5.2.4

6.6.2.2 Entkuppler schalten

EV_RS232_streckenbefehl.Entkoppler.0 = Entkuppler-Position

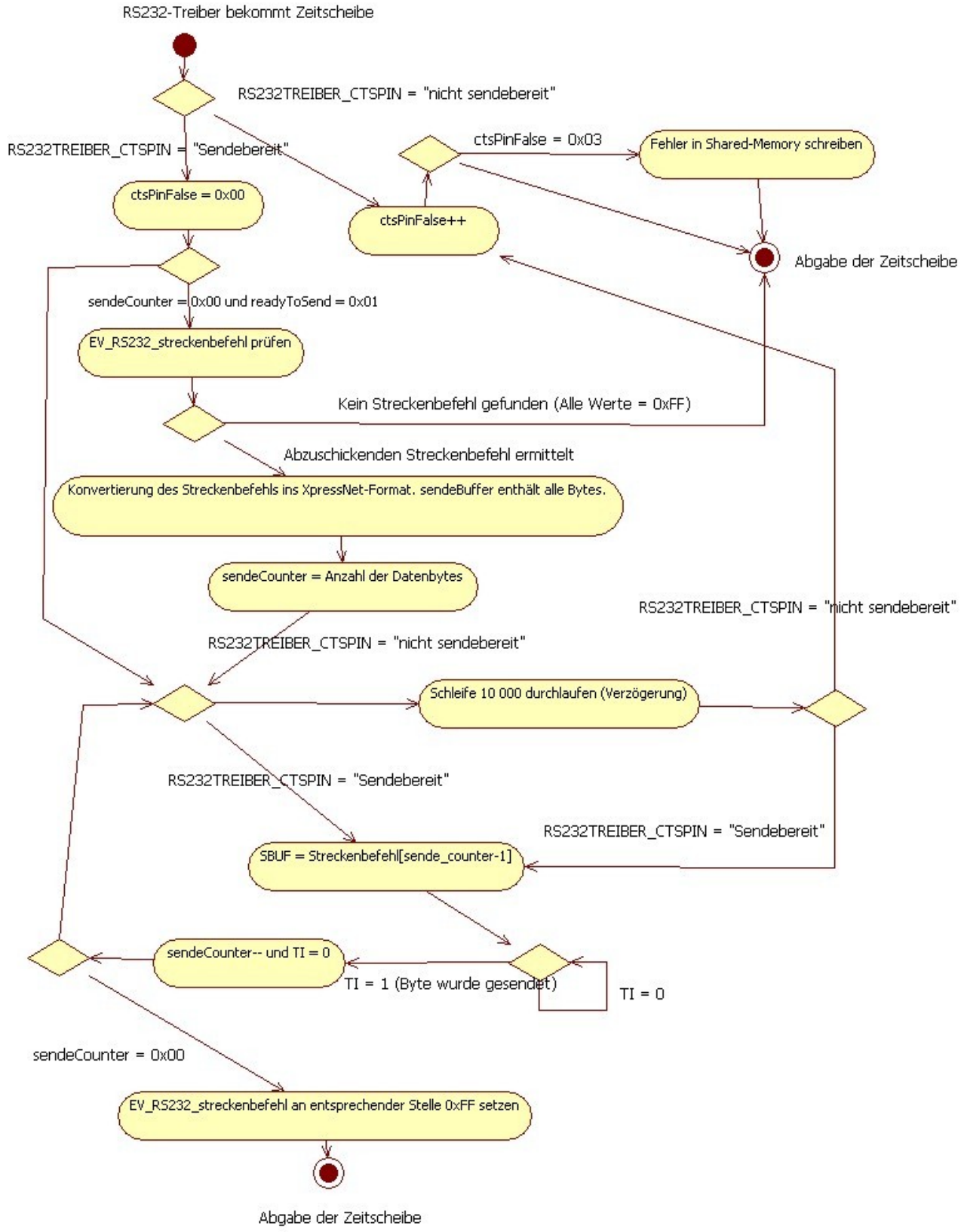
- xxxx xxx0 → Anheben
- xxxx xxx1 → Senken

Das Senken geschieht automatisch.

Um einen Entkuppler zu heben, muss der entsprechende Ausgang zunächst deaktiviert und dann wieder aktiviert werden. Bei der Erzeugung des Entkuppel-Befehls (aktivieren), muss überprüft werden ob im sendeBuffer bereits ein Deaktivieren-Befehl der gleichen Adresse und des gleiches Ausgangs vorliegt. Ist das nicht der Fall wird ein Solcher zuerst generiert.

EV_RS232_streckenbefehl.Entkoppler.7-1 = Entkuppler-Adresse

Die Erzeugung des XpressNet-Befehls geschieht so wie in 6.5.2.2 und 5.2.4 beschrieben.


RS232-Treiber bekommt Zeitscheibe
RS232TREIBER_CTSPIN = "nicht sendebereit"
RS232TREIBER_CTSPIN = "Sendebereit"
ctsPinFalse = 0x03
Fehler in Shared-Memory schreiben
ctsPinFalse = 0x00
ctsPinFalse++
Abgabe der Zeitscheibe
sendeCounter = 0x00 und readyToSend = 0x01
EV_RS232_streckenbefehl prüfen
Kein Streckenbefehl gefunden (Alle Werte = 0xFF)
Abzuschickenden Streckenbefehl ermittelt
Konvertierung des Streckenbefehls ins XpressNet-Format. sendeBuffer enthält alle Bytes.
sendeCounter = Anzahl der Datenbytes
RS232TREIBER_CTSPIN = "nicht sendebereit"
RS232TREIBER_CTSPIN = "nicht sendebereit"
Schleife 10 000 durchlaufen (Verzögerung)
RS232TREIBER_CTSPIN = "Sendebereit"
RS232TREIBER_CTSPIN = "Sendebereit"
SBUF = Streckenbefehl[sende_counter-1]
sendeCounter-- und TI = 0
TI = 1 (Byte wurde gesendet)
TI = 0
sendeCounter = 0x00
EV_RS232_streckenbefehl an entsprechender Stelle 0xFF setzen
Abgabe der Zeitscheibe

7 C Implementierungsdetails

7.1 Interrupt-Service-Routinen

```
DataReceived () interrupt 4
{
//Ueberpruefen welcher Interrupt ausgelöst wurde (Sende oder Empfang)
 if(RI == 1)
{
```

1. Sendebestätigung vom Lenz-Modul abarbeiten
    1. Bereits ein Antwortbyte vom Modul erhalten wurde
        1. Empfangspuffer prüfen ob gültiges Byte[1] oder Byte[2] empfangen wurde
            1. Bei Fehler Fehlervariable inkrementieren
    2. Kein Antwortbyte erhalten wurde
        1. Empfangspuffer auf Antwortbyte prüfen und in Array schreiben
    3. Alle Antwortbytes erhalten
        1. Falls vorhanden nächsten Fahrbefehl versenden
2. RI Interrupt zurücksetzen

```
}
else {
```

1. TI Interrupt zurücksetzen

```
} }
```

Die 4 ergibt sich anhand folgender Berechnung (Quelle: C.pdf, s.18):

Interruptnummer = (Interruptadresse(dezimal) – 3) / 8

Sowohl das C.pdf als auch das User Manual weisen eine Tabelle mit allen Interruptadressen auf.

8 Probleme

9 Anhang