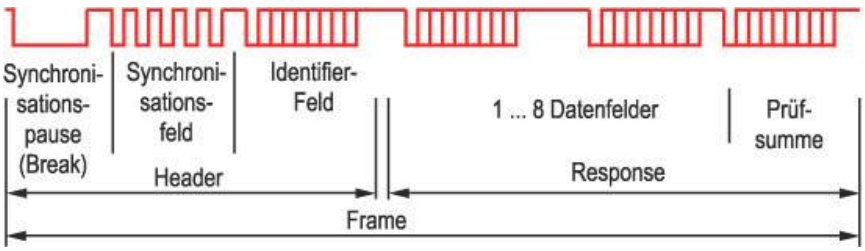
Implementierung eines LIN-Slaves



Projektarbeit
Bus-Syteme und Interfaces

NTA FH Isny
12. Info

Matthias Müller - David Mayr - Martin Junginger

Inhaltsverzeichnis

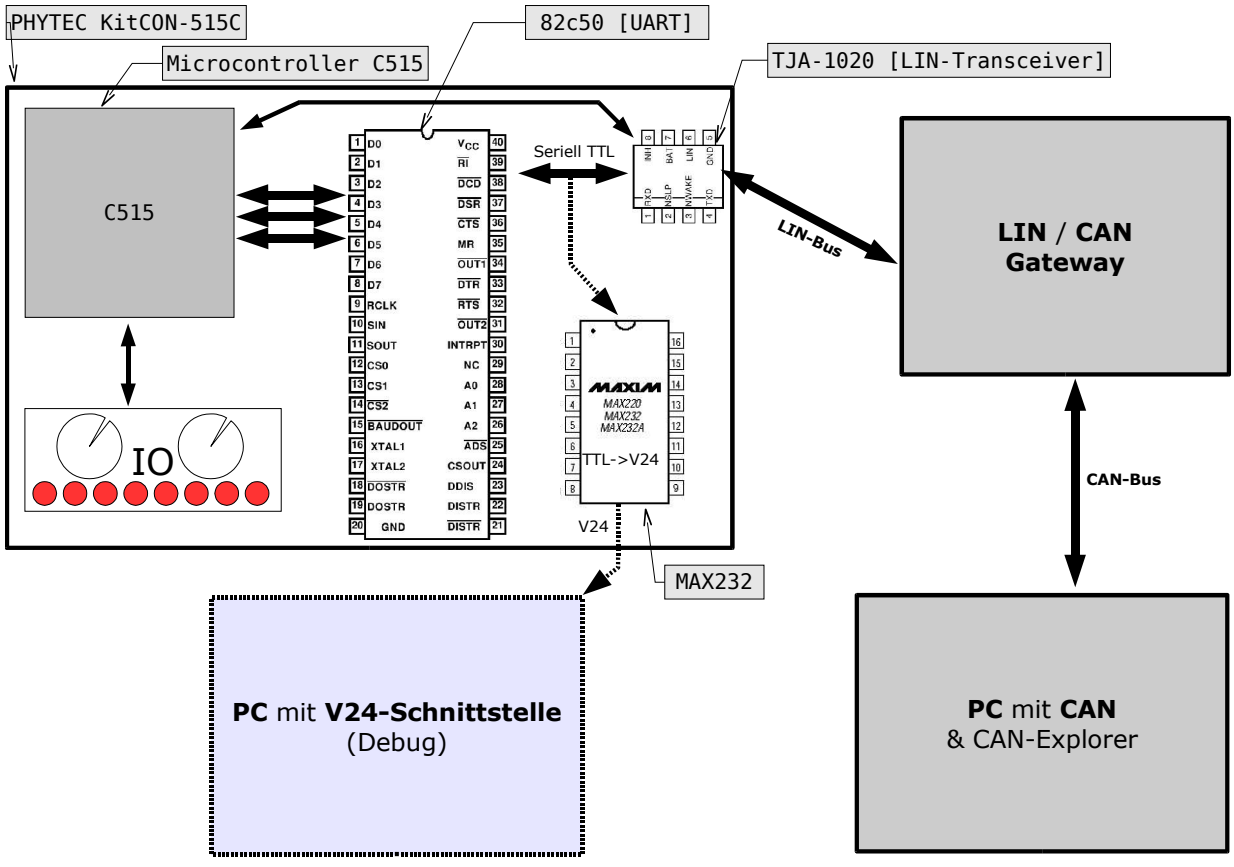
1 Einleitung

In diesem Projekt soll ein LIN-Slave mit Hilfe eines C515-Microcontrollers auf einem Phytec-Entwicklerboard realisiert werden. Dabei kommen außer dem Entwicklerboard noch der LIN-Transceiver TJA1020 und ein Standard-UART 82c50 zum Einsatz.
Mit diesem LIN-Slave soll nun über ein CAN/LIN-Gateway hinweg mit einem CAN-Knoten (PC) kommuniziert bzw. einfaches digitales IO betrieben werden können, so daß z.B. die LEDs auf dem Entwicklerboard per PC-Software angesteuert und Jumperstellungen (oder auch Werte der A/D-Wandler) am PC als CAN-Knoten erfasst werden können.

2 Hardware

Wir verwenden das Entwicklerboard KitCON-515C von Phytec. Auf der Platine im Euro-Format[1] ist sind mehr als ein drittel der Fläche frei und mit einem Lochraster versehen auf dem die zusätzlichen Bauteile Platz finden.
Da die einzige serielle Schnittstelle des C515-Microcontrollers zum Software-Upload und Debuggen mit der Keil-Entwicklungsumgebung im Einsatz ist, kann mit ihr nicht mehr der LIN-Transceiver angesteuert werden. Anstelle einer Software-Implementierung der nötigen zweiten seriellen Schnittstelle über zwei IO-Pins, was die eine Alternative gewesen wäre, haben wir uns dazu entschieden den weit verbreiteten UART 82c50 als Hardware-Lösung einzusetzen.

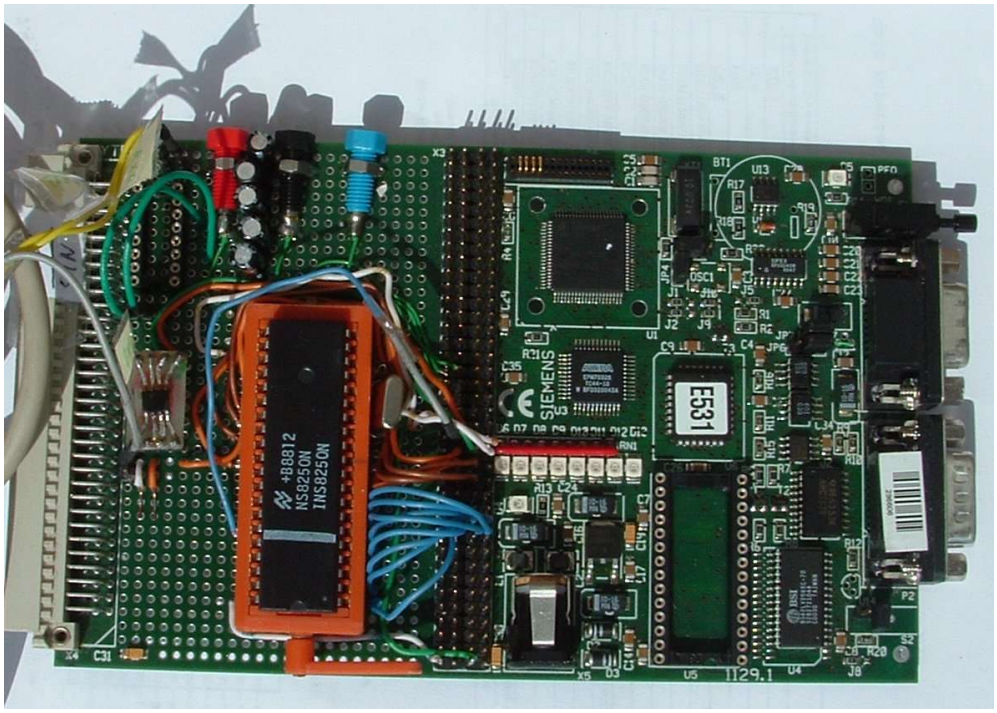
Der Grundlegende Aufbau sieht wie folgt aus:

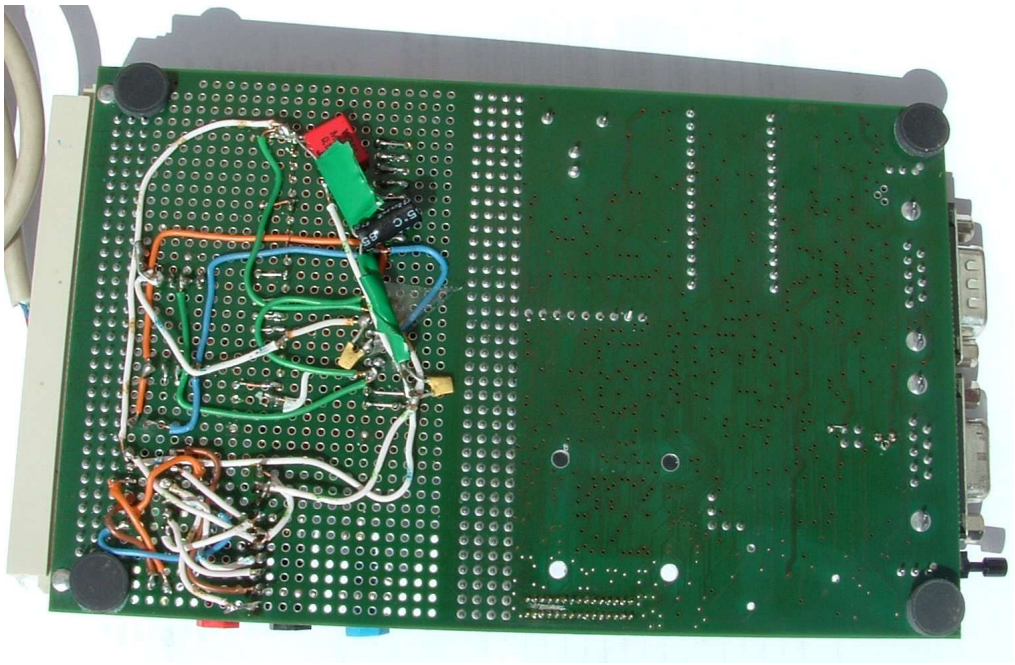


1 100 x 160 mm

2.1 Phytec Entwicklerboard KitCON-515C

Auf dem von uns verwendeten Entwicklerboard KitCON-515C arbeitet ein Infineon C515 Microcontroller. Das Board bietet auch schon 8 LEDs an, die über den IO-Port 4 zur Ausgabe genutzt werden können.





2.2 UART 82c50

Beim UART 82c50 handelt es sich um einen programmierbaren seriellen Receiver/Transmitter Baustein, dessen Baudrate programmierbar ist. Für unser Projekt haben wir uns für die PDIP-Ausführung des Bausteins entschieden, weil diese besser auf dem Entwicklungsboard KitCON-515C unterzubringen war. Die für unser Projekt benötigte Funktionalität des UART haben wir in einer entsprechenden Datei UART-82c50.c untergebracht. Dort befinden sich alle Low-Level-Funktionen, die für die Ansteuerung des LIN-Transceivers über den 82c50 von Nöten sind. Diese Funktionen sind im Abschnitt Low-Level-Funktionen näher beschrieben.

Wir haben im einzelnen folgende Pins und Register des Bausteins in unseren Funktionen bzw. im Modul verwendet:

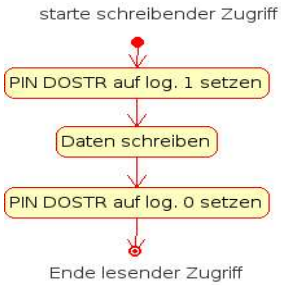
2.2.1 Verwendete Pins

DISTR:

Der Pin DISTR (Data In Strobe) wird benötigt, um Daten vom 82c50 auf den Mikrocontroller zu übertragen. Der Pin muss vor einem lesenden Zugriff des Microcontrollers auf den 82c50 auf log. 1 gesetzt werden. Sobald der Pin gesetzt ist können Daten vom 82c50 gelesen werden. Achtung - vor dem lesenden Zugriff auf den 82c50 muss vom Mikrocontroller an den Daten-Pins des 82c50 die Maske 0xFF angelegt werden. Wird dies nicht gemacht, so ist ein Lesen vom 82c50 fehlerhaft. Nach erfolgreichem Lesen vom 82c50 muss der Pin DISTR wieder auf log. 0 zurückgesetzt werden.

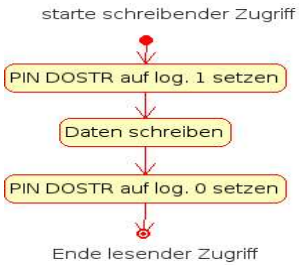
DOSTR:

Pin DOSTR (Data Out Strobe) wird benötigt, um die Daten auf den 82c50 zu übertragen. Vor dem Übertragen der Daten des Microcontrollers in das jeweilige Register des 82c50 muss der Pin DOSTR auf log. 1 gesetzt werden. Wird dies nicht gemacht, kann nicht auf das zuvor ausgewählte Register des 82c50 geschrieben werden. Nach erfolgreichem Schreiben muss der Pin wieder auf log. 0 gesetzt werden.



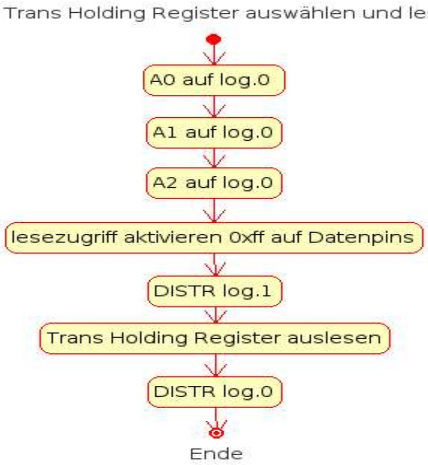
D0-D7:

Die Pins D0-D7 (Daten Bits 0-7) sind die Daten Ein- und Ausgänge des 82c50. Sie bilden bei uns die Schnittstelle zwischen Mikrocontroller und serieller Schnittstelle. Alle Informationen (Registerinhalte, zu versendende/empfangende Daten) die der 82c50 erhält/bereitstellt werden mit Hilfe dieser 8 Pins an den 82c50 übertragen oder von ihm gelesen. D0 ist das niederwertige Bit (LSB) und D7 ist das höchstwertige Bit (MSB).



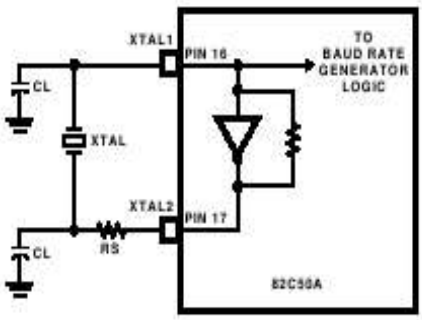
A0, A1, A2:

Diese drei Adress-Pins (Register Select Pin's) werden dazu verwendet um die Adresse des internen Registers des 82c50 zu bestimmen.



XTAL1 / XTAL2:

Diese beiden Pins (Crystal/Clock) werden für den internen Baud Rate Generator benötigt. An XTAL1 kann ein externer Takt in den Baustein geführt werden, wenn kein eigener Quarz verwendet werden soll. Sollte an eigener Quarz verwendet werden, dann muss dieser an XTAL1 und XTAL2 angeschlossen werden. Wir haben im Rahmen unseres Projektes einen eigenen Quarz an XTAL1 und XTAL2 angeschlossen der mit 1,8 MHz taktet.



SOUT:

Der Pin SOUT (Serial Data Out) ist der serielle Datenausgang des 82c50. Wir haben diesen Pin mit dem seriellen Eingang des TJA1020 verbunden. Außerdem besteht noch eine Verbindung zum seriellen Eingang des MAX232 Pegelwandlers, den wir zum Debuggen dazu gebaut haben – dadurch können wir mit Hilfe eines PCs mit serieller Schnittstelle einfach den Datenverkehr beobachten und analysieren. Die Auswahl, welcher Baustein nun das Signal an SOUT erhalten soll, haben wir mit Hilfe eines Jumpers auf unserer Platine gelöst.

GND:

An Pin GND wird die gemeinsame Masse angelegt. Wir greifen unsere Masse an einem Masse Port des Microcontrollers ab.

/BAUDOUT:

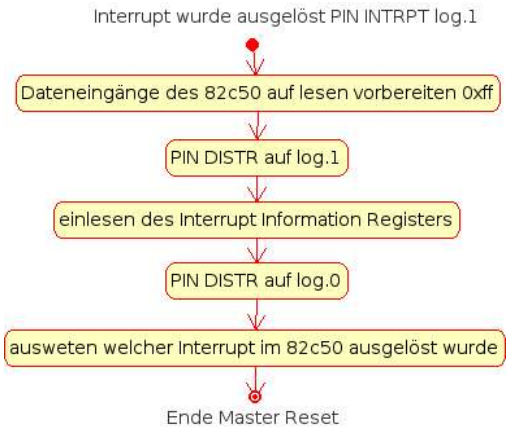
An Pin BAUDOUT wird der mit Hilfe der Register DLL und DLM erzeugte Takt aus dem Baustein herausgeführt um in dann am Pin RCLK wieder in den Baustein einzuführen. Sollte ein anderes Gerät auch den im Baustein eingestellten Takt benötigen, so kann dieser gegebenenfalls auch an diesem Pin abgegriffen werden.

MR:

Mit Hilfe des MR-Pins (Master Reset) kann auf dem Baustein ein Reset durchgeführt werden. Wir haben uns im Rahmen unserer Projektarbeit darauf geeinigt, dass wir den MR-Pin mit Hilfe des Mikrocontrollers ansteuern – dadurch kann dieser jederzeit durch das Programm durchgeführt werden.

INTRPT:

Der Pin INTRPT (Interrupt Request) dient zum melden eines Interrupts an den Mikrocontroller. Sobald dieser Pin den Zustand log. 1 annimmt ist ein Interrupt auf dem UART 82c50 aufgetreten der aufgetretene Interrupt muss dann durch den Mikrocontroller ausgewertet werden. Dazu muss das Interrupt Identifikation Register (IIR) des 82c50 ausgelesen und gedeutet werden. Wir haben diesen Pin an einen externen Interrupt des Mikrocontrollers angeschlossen um somit die Funktionalität zu garantieren.



SIN:

Der Pin SIN (Serial Data In) dient als serieller Dateneingang des 82c50. Wir haben diesen Pin mit dem seriellen Ausgang des TJA1020 verbunden um Daten, die über diesen Transceiver empfangen werden, seriell an den 82c50 zu übertragen. Diese Daten stehen dann wiederum dem Mikrocontroller zur Verfügung. Desweiteren wurde der Pin SIN wie schon erwähnt aus Debuggründen mit dem seriellen Ausgang des MAX232 Bausteins verbunden.

VCC:

An diesem Pin wird die positive Spannung von 5V angelegt, um den 82c50 zu betreiben. Diese Spannung wird an einem Pin des Phytec-Boards abgegriffen.

Hinweis: *Wir haben zwischen VCC und GND zwei Kondensatoren parallel geschaltet um eventuelle Spannungsschwankungen abzufangen.*

CS0-/CS2:

Diese drei Pins (Chip Select) dienen als Eingänge um die Lese-/Schreib-Eingangssignale (DISTR / DOSTR) des Bausteins zu aktivieren – und damit den Baustein selbst zu aktivieren. Sollten diese Pins nicht korrekt beschaltet sein dann ist das Arbeiten mit dem 82c50 nicht möglich. Darum haben wir diese drei Eingänge fest verdrahtet, da wir ohnehin nur mit diesem seriellen Baustein und nicht mit mehreren arbeiten. Pin CS0, CS1 wurden mit VCC, und Pin /CS2 mit GND verbunden. Ob der Baustein ausgewählt wurde kann an Pin CSOUT gemessen werden. Der Pegel des Pins CSOUT sollte bei erfolgreicher Auswahl des Bausteins auf log. 1 sein.

CSOUT:

Wie eben erwähnt ist der Pin CSOUT (CHIP Selected Out) ein Ausgang, an dem festgestellt werden kann, ob der Chip zum Betrieb ausgewählt wurde. Da es sich bei diesem Pin um einen Ausgang zur Kontrolle handelt, haben wir uns dazu entschlossen diesen nicht direkt mit dem Mikrocontroller zu verbinden, zumal wir nicht über ausreichend I/O-Ports verfügen und der Chip fest ausgewählt wurde.

/ADS:

Der Pin ADS (Adress Strobe) kann verwendet werden, wenn im Multiplex-Mode gearbeitet werden soll. Wir haben nur einen 82c50 und haben deswegen den /ADS Pin auf log.0 gelegt.

RCLK:

Der Pin RCLK dient als Eingang. An diesem Eingang kann der Takt von Pin /BAUDOUT oder von einem anderen externen Taktgenerator an angelegt werden, wird also benötigt um die Empfangs-Einheit mit der ausgewählten Baudrate zu betreiben. Wir haben uns dafür entschieden /BAUDOUT als Taktquelle zu wählen und die beiden Pins miteinander verbunden.

Bemerkung:

Alle hier nicht aufgeführten Pins haben keinen weiteren Einfluß und wurden offen gelassen.
Die genaue Verdrahtung der einzelnen Pins ist dem Schaubild zu entnehmen.

2.2.2 Verwendete Register

Auflistung der verwendeten internen Register und ihre dazugehörigen Adressen:

DLAB	A2	A1	A0	Kürzel	Register
0	0	0	0	RBR	Receiver Buffer Register
0	0	0	0	THR	Transmitter Holding Register
0	0	0	1	IER	Interrupt Enable Register
X	0	1	0	IIR	Interrupt Identification Register
X	0	1	1	LCR	Line Control Register
X	1	0	1	LSR	Line Status Register
1	0	0	0	DLL	Divisor Latch (LSB)
1	0	0	1	DLM	Divisor Latch (MSB)

Receive Buffer Register (RBR):

Das Empfangs-Register kann für 5,6,7 oder 8 Datenbits pro Zeichen eingestellt werden. Das erste empfangene Datenbit (LSB) ist RBR(0). Das Lesen des Empfangspuffers wird bei uns durch eine Interrupt-Routine erledigt.

Transmit Holding Register (THR):

Dieses Register beinhaltet das Daten-Byte, das als nächstes seriell versendet werden soll.

Interrupt Enable Register (IER):

Im Interrupt Enable Register können die zu verwendeten Interrupts ausgewählt werden. Bit0-Bit3 sind hierbei relevant. Bit4-Bit7 sind log.0 und fallen aus der Betrachtung. Wir verwenden im Rahmen des Projektes die Interrupts IER (0-2).

Register	Bit	Bedeutung
IER	0	Wenn IER(0)= log. 1 ist, dann wird der Received Data Available Interrupt aktiviert bei log.0 ist er deaktiviert.
IER	1	Wenn IER(1= log. 1 ist, dann wird der Transmitter Holding Register Empty Interrupt aktiviert bei log. 0 ist er deaktiviert.
IER	2	Wenn IER(1= log. 1 ist, dann wird der Receiver Line Status Interrupt aktiviert bei log. 0 ist er deaktiviert.
IER	3	Wenn IER(1= log. 1 ist, dann wird der Modem Status Interrupt aktiviert bei log. 0 ist er deaktiviert.
IER	1-4	keine Bedeutung

Interrupt Identification Register (IIR):

Mit Hilfe des Interrupt Identification Registers kann man den durch den 82c50 ausgelösten Interrupt bestimmen. Zur Bestimmung des ausgelösten Interrupts müssen Bit0-Bit2 ausgewertet werden. Folgende Tabelle kann zur Auswertung benutzt werden:

Bit2	Bit1	Bit0	Priorität	Interrupt Flag	Interrupt Quelle	Durch den Interrupt veränderte Register
X	X	1		Keins	Keine	
1	1	0	1	Receiver Line Status	OE, Pe, FE oder BI	LSR read
1	0	0	2	Received Data Available	Receiver Data Available	RBR read
0	1	0	3	THRE	THRE	THRE oder THR write ist die Interrupt-Quelle

Achtung: X = Nicht Definiert (egal ob 0 oder 1)

In unserer Interrupt Service Routine werden alle drei Interrupts ausgewertet.

Line Control Register (LCR):

Das Format der Datenzeichen wird durch dieses Register beeinflusst. Auf welche Art und Weise dies geschehen kann, wird in folgender Tabelle verdeutlicht:

Register	Bit	Bedeutung
LCR	0 1	**Word Length Select:** Gibt die Anzahl der Bits in jedem empfangenen oder versendeten seriellen Zeichen an.

LCR(1)	LCR(2)	Zeichenlänge
0	0	5 Bit's
0	1	6 Bit's
1	0	7 Bit's
1	1	8 Bit's

Register	Bit	Bedeutung
LCR	2	**Stop Bit Select:** Gibt die Anzahl der zu verwendeten Stopbits bei jedem zu versendenden Zeichen an.

Zustand	Zeichnlänge	Bedeutung
Log. 0	>5	Es wird ein Stopbit verwendet
Log. 1	5	Es werden 1.5 Stopbit's verwendet
Log. 1	>5	Es werden 2 Stopbits verwendet

Register	Bit	Bedeutung
LCR	3	**Parity Enable:** Einschalten von Parity Prüfung

Zustand	Bedeutung
Log. 0	Parity ist deaktivirt
Log. 1	Ein Parity Bit wird zwischen dem letzten Datenzeichen und dem Stopbit(s) eingesetzt und geprüft.

Register	Bit	Bedeutung
LCR	4	**Even Parity Select:** Even Parity auswählen wenn Parity aktiviert ist

Zustand	Bedeutung
Log. 0	Wählt ungerade Parity
Log. 1	Wählt gerade Parity

Register	Bit	Bedeutung
LCR	5	**Stick Parity:** Wen Parity aktiviert ist, dann kann dieses Bit dazu verwendet werden um das Parity Bit auf einen gewünschten zustand zu zwingen. (log. 1)

Zustand	Bedeutung
Log. 0	Stick Parity ist deaktiviert
Log. 1	Stick Parity ist aktiviert

Register	Bit	Bedeutung
LCR	6	**Break Control:** Wird dazu verwendet um den seriellen Ausgang SOUT zu manipulieren

Zustand	Bedeutung
Log. 0	Break Control ist deaktiviert
Log. 1	Der serielle Ausgang wird auf log. 0 gesetzt

Register	Bit	Bedeutung
LCR	7	**Divisor Latch Access Bit(DLAB):** Dieses Bit wird dazu verwendet um den Adressbereich von A2,A1,A0 zu vergrößern um damit das Divisor Latch Register auswählen zu können.

Zustand	Bedeutung
Log. 0	Das Bit DLAB ist log. 0
Log. 1	Das Bit DLAB ist log. 1

Wir arbeiten im Projekt mit 0x03h als Maske für das LCR Register - haben also 8 Datenbits, 1 Stopbit und keine Parität ausgewählt.

Line Status Register (LSR):

Das Line Status Register enthält Statusinformationen über serielle Verbindung. Die einzelnen Bits werden in folgender Tabelle aufgeschlüsselt.

Register	Bit	Register Name	Kürzel	log. 1	log. 0
LSR	0	Data Ready	DR	Ready	Not Ready
LSR	1	Overrun Error	OE	Error	No Error
LSR	2	Parity Error	PE	Error	No Error
LSR	3	Framing Errror	FE	Error	No Error
LSR	4	Break Interrupt	BI	Break	No Break
LSR	5	Transmitter Holding Register Empty	THRE	Empty	Not Empty
LSR	6	Transmitter Empty	TEMT	Empty	Not Empty
LSR	7	Not Used	-	-	-

Register	Bit	Bedeutung
LSR	0	**Data Ready:** Zeigt an ob neue Daten empfangen wurden

Zustand	Bedeutung
Log. 0	Empfangspuffer wurde von der CPU gelesen
Log. 1	Ein neues Zeichen wurde empfangen

Register	Bit	Bedeutung
LSR	1	**Overrun Error:** Dieses Bit zeigt an ob ein Overrun Error aufgetreten ist. Also ob die CPU das empfangsregister rechtzeitig, vor dem empfang eines neuen Zeichens, gelesen hat

Zustand	Bedeutung
Log. 0	Sobald die CPU den Empfangspuffer gelesen hat wird das Bit gelöscht
Log. 1	Der CPU hat den Empfangspuffer noch nicht gelesen

Register	Bit	Bedeutung
LSR	2	**Parity Error:** Ein Parity Error tritt auf, wenn das empfangene Zeichen keinen korrekten even oder odd Parity besitzt. (Diese Einstellungen werden im LCR Register vorgenommen)

Zustand	Bedeutung
Log. 0	LSR Register wurde von der CPU gelesen
Log. 1	Es ist ein Parity Error aufgetreten

Register	Bit	Bedeutung
LSR	3	**Framing Error:** Dieser Error trit auf wenn das empfangene Zeichen kein gültiges Stopbit aufweist.

Zustand	Bedeutung
Log. 0	LSR Register wurde von der CPU gelesen
Log. 1	Ein Framing error ist aufgetreten (Stopbit und oder Paritybit = log. 0)

Register	Bit	Bedeutung
LSR	4	**Break Interrupt:** Dieses Bit wird gesetzt wenn der empfangs Daten Eingang den Zustand log. 0 länger als eine f*ull word Transmission* Zeit(startbit + Datenbits + parity + stopbits) hält.

Zustand	Bedeutung
Log. 0	LSR Register wurde von der CPU gelesen
Log. 1	Der empfangs Daten Eingang hält den Zustand log. 0 länger als eine f*ull word Transmission* (startbit + Datenbits + parity + stopbits) also eine volle Frame übertragung lang

Register	Bit	Bedeutung
LSR	5	**Transmit Holding Register Empty:** Dieses Bit bedeutet, dass der 82c50 bereit ist ein weiterers Zeichen zu senden.

Zustand	Bedeutung
Log. 0	Laden von Daten ins Trans Holding Register
Log. 1	Ein Zeichen wurde vom Trans Holding Register ins Transmit shift register übertragen. Bereit für neue Daten. Dieses Bit wird nocht gelöscht wenn das LSR ausgelesen wird.

Register	Bit	Bedeutung
LSR	6	Transmit Empty(TEMT): Dieses Bit signalisiert das Trans Holding Register und Transmitter Shift Register leer sind.

Zustand	Bedeutung
Log. 0	Ein Zeichen befindet sich im THR
Log. 1	THR und TSR sind leer

Register	Bit	Bedeutung
LSR	7	Dieses Bit wird nicht verwendet und ist log. 0

Divisor Latch (DLAB):

Zwei 8 Bit Divisor Latch Register speichern den Baudraten-Divisor in einem 16 Bit Binary

Format. Mit Hilfe des Divisors wird dann die gewünschte Baudrate erzeugt.

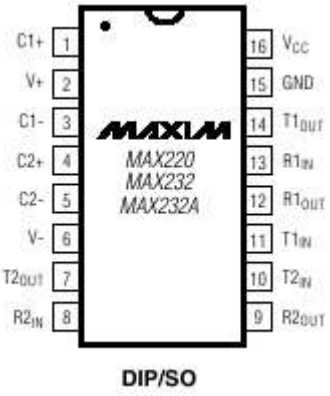
Beispiel: Gesuchte Baudrate 9600 Baud
Verwendete Frequenz 1,8432 MHz

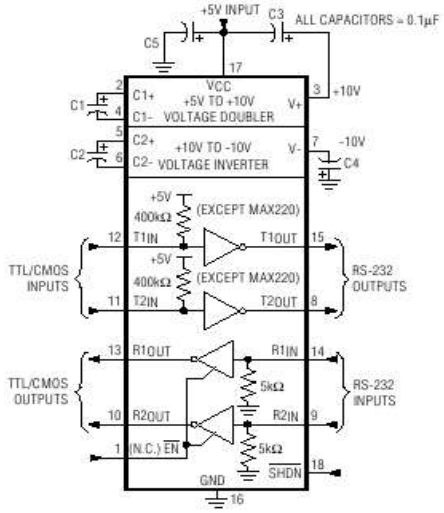
Divisor: 18432/(9600*16)
Divisor: =12 -> 0C Hex ->DLL 00001100
->DLM 00000000

Wir benutzen 9600 Baud zur Übertragung.
Diese 9600 Baud werden mit der Funktion confUART() eingestellt.

2.3 MAX232-Pegelwandler

Beim MAX232 handelt es sich um einen V24 Pegelwandler, welcher TTL Signale in V24 wandelt und/oder V24 auf TTL umsetzt. Es könne bis zu zwei serielle Schnittstellen angeschlossen werden. Um den MAX232 betreiben zu können, mussten wir 5 Kondensatoren anbringen. Wir haben 5 * 1,0 µF Kondensatoren auf der Platine verbaut. Die folgende Skizze veranschaulicht das Ganze:





Wir haben folgende Pins des MAX232 verwendet:

- C1+, C1-, C2+. C2-, V-, V+
  - An diesen Pins wurden Kondensatoren angebracht — Pin 2,3,4,5,6,7

- T1_out, T1_in
  - T1_in ist die Datenleitung vom seriellen Baustein zum zum MAX232 — Pin 12
  - T1_out ist die Datenleitung vom Max232 zum Gegenüber (PC) — Pin 15

- R1_out, R1_in
  - R1_in ist die Datenleitung vom Gegenüber (PC) zum Max232 — Pin 14
  - R1_out ist die Datenleitung vom Max232 zum seriellen Baustein — Pin 13

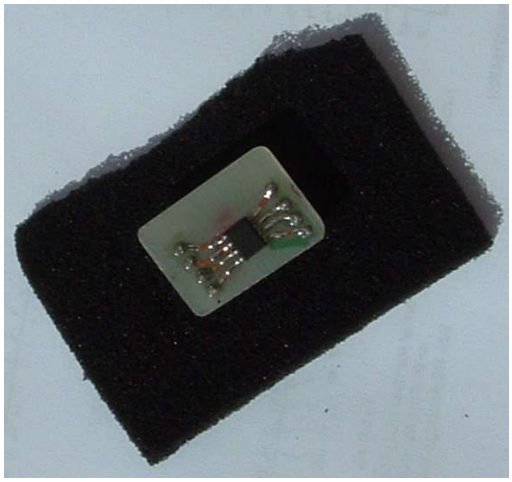
- VCC Versorgungsspannung 5V
- GND gemeinsame Masse

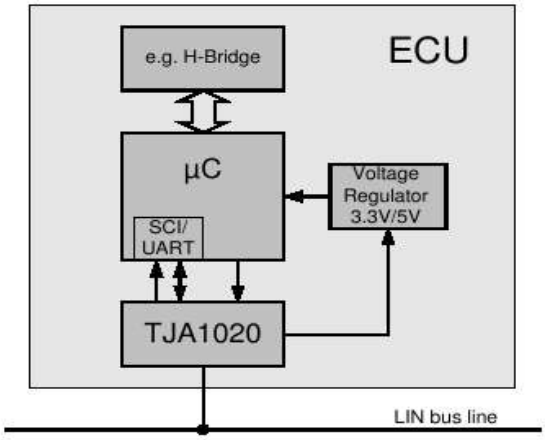
Alle hier nicht erwähnten Pins sind offen gelassen worden.

Wir haben den Max232 dazu verwendet, um uns Debug Informationen per Hyperterminal auf einem anderen Rechner anzeigen zu lassen. Die für die Verbindung notwendigen Pins greifen wir direkt, an dafür vorgesehenen Pins an der Platine ab.

2.4 LIN-Transceiver TJA1020

Der TJA1020 ist ein Low Power LIN-Transceiver. Er unterstützt die Bus Signale die im LIN Protokoll Verwendung finden. Local Interconnect Network (LIN) ist ein serielles Bus-Protokoll welches für das kontrollierte versenden von Daten über eine Eindrahtleitung verwendet wird. Im Rahmen des Projektes verwenden wir den LIN-Transceiver, um mit dem LIN-Master (ein CAN/LIN-Gateway) Daten auszutauschen. Den genauen Schaltungsaufbau kann man dem Bild unten entnehmen.
Um den SMD-Baustein besser auf dem Lochraster des Phytec-Boards verdrahten zu können, haben wir uns eine kleine Adapter-Platine geätzt und gelötet.





2.4.1 Betriebsmodi

Der TJA1020 ist ein Baustein der 4 Zustände kennt.

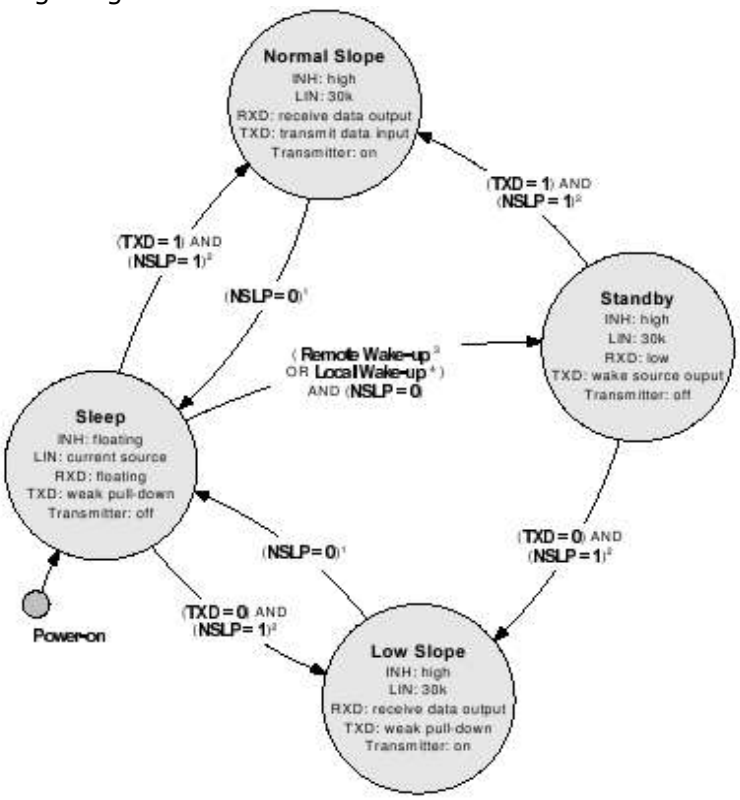
- Normal Slope Mode
- Sleep Mode
- Standby Mode
- Low Slope Mode

Eine Beschreibung der 4 Modi kann der Tabelle entnommen werden.

Mode	NSLP	TXD	RXD	INH	Transmitter
Sleep	0	Weak-Pull-Down	Floating	Floating	off
Standby	0	Weak-Pull-Down wenn remote Wake-Up;strong-Pull-Down wenn lokaler Wake-Up	Low	High (Vbat)	off

Mode	NSLP	TXD	RXD	INH	Transmitter
Low Slope	1	Weak-Pull-Down	High: reccessive state Low: dominate state	High (Vbat)	on
Normal Slope	1	Weak-Pull-Down	High: reccessive state Low: dominant state	High (Vbat)	on

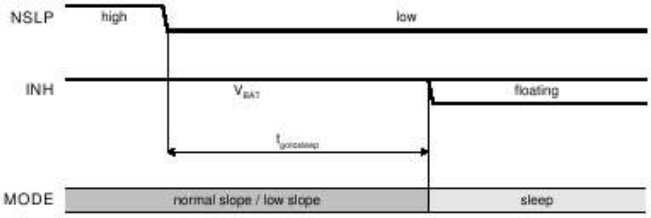
Folgendes Diagramm zeigt wie der TJA1020 programmiert werden muss um in den gewünschten Modus zu gelangen:



Sleep Mode:

Im Sleep Modus nimmt der TJA1020 nur sehr wenig Leistung auf. Der TJA geht nach dem Anschalten automatisch in den Sleep Modus und wartet bis er geweckt wird. Der Pin INH ist, während sich der TJA1020 im Sleep Mode befindet, auf log. 0, aber sobald dieser aufgeweckt wird wechselt Pin INH auf log.1. Das Wecken des Bausteins kann entweder lokal durchgeführt werden oder durch den Master, der ein Wake-Up Signal schickt. Nach dem aufwecken des Bausteins geht dieser automatisch in den Standby Mode über. Um in den Sleep Mode zu wechseln muss eine log. 0 an den Sleep Kontroll Eingang (Pin NSLP) für die Zeit >Tgotosleep (2-5 µs) angelegt werden. Folgendes Timing-Diagramm verdeutlicht dies:

Sleep Mode Timing:

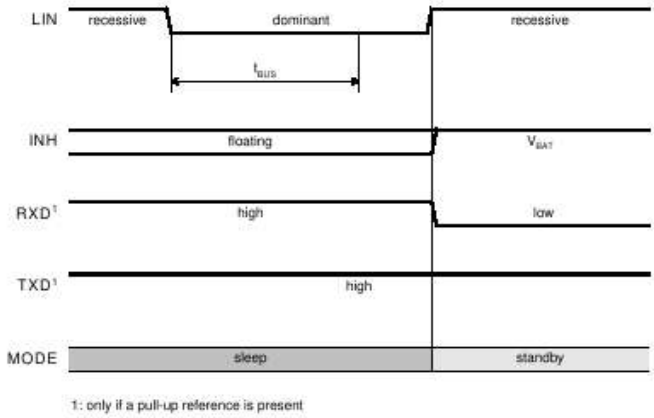


Standby Mode:

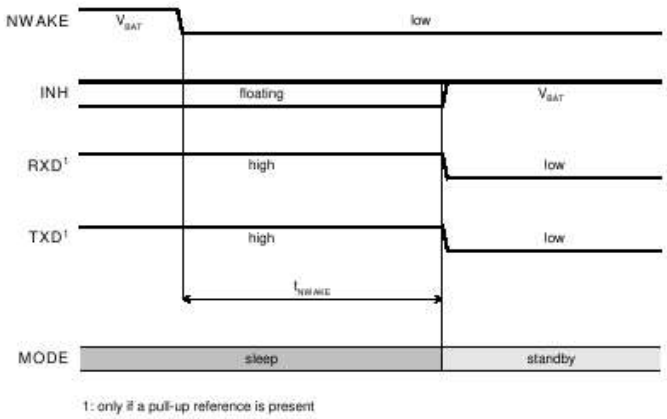
Der Standby Mode ist ein Modus in den der TJA1020 automatisch nach einem localen -oder remote-Wake-Up übergeht. Sobald der TJA im Standby Mode ist liegt an Pin INH ein von Vbat abhängende Spannung an. Der TJA signalisiert seinen Zustand durch ein low level an Pin RXD. Dieses Signal kann als Wake-Up-interrupt für den Microcontroller verwendet werden. Im Pojekt haben wir dies nicht berücksichtigt, da unser Microcontroller nicht ausgeschaltet werden sollte. Das locale aufwachevent wird durch ein Pull-Down an Pin TXD eingeleitet. Ein remote-Wake-Up verursacht einen weak-Pull-Down. Wie in den beiden Diagrammen zu sehen ist.

Standby Mode Timing mittels remote-Wake-Up:

Wie man aus der Grafik erkennen kann wird ein Remote Wake-Up Event durch einen dominanten Pegel am LIN-Pin erreicht, der für die Zeit Tbus anliegen muss und von einer rezessiven Flanke gefolgt wird.



Standby Mode Timing Local Wake-Up:



Eine negative Flanke an Pin NWAKE für die Zeit Tnwake initiiert das Local Wake-Up Event. Sobald der Standby Mode erreicht ist muss an Pin TXD und RXD ein Low Pegel zu messen sein.

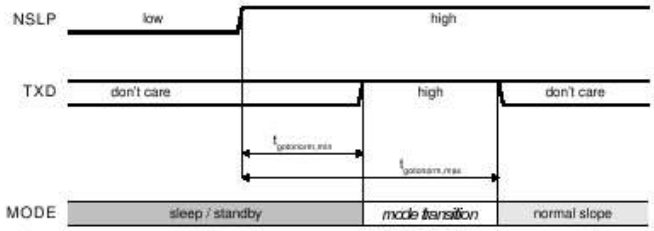
Normal Slope Mode:

Der Normal Slope Mode wird dazu verwendet um Daten über den LIN-Bus zu versenden und zu empfangen. Der Bus Datenstrom wird vom Empfänger (Receiver Unit) in einen digitalen Bit-Strom umgewandelt und an Pin RXD an den Mikrocontroller weitergereicht. Ein High Level an Pin RXD repräsentiert ein Rezessiven Level auf dem LIN-Bus; ein Low Level an Pin

RXD repräsentiert hingegen einen dominierenden LIN-Bus Level. Die Sendeeinheit des TJA1020 wandelt die Daten vom Mikrocontroller an Pin TXD in ein für den LIN-Bus passenden Datenstrom um, welcher dann über den Bus versendet wird. Ein Low-Level am TXD Eingang zeigt einen dominierenden LIN-Bus-Level und ein High-Level an Pin TXD zeigt einen rezessiven LIN-Bus Pegel an.

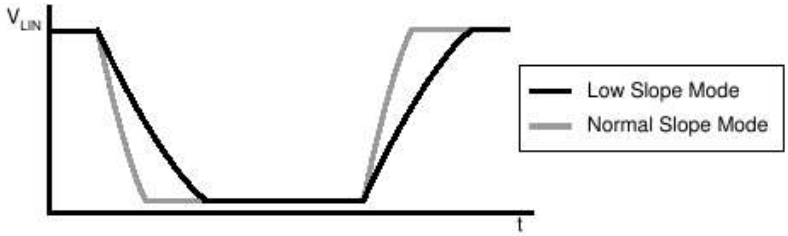
Im Normal Slope Mode schaltet der Interne Slave Terminierungs-Widerstand (Rslave) den LIN-Bus Pin high. Pin INH wird ebenfalls high.

In den Normal Slope Mode kann durch das Setzen von Pin NSLP und TXD auf High, für die Zeit Tgotonorm, aus dem Sleep oder Standby Mode gewechselt werden. Folgendes Schaublid verdeutlicht dies:



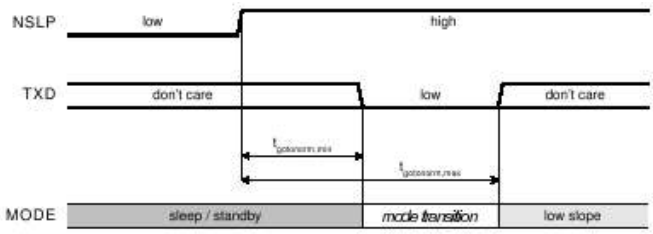
Low Slope Mode:

In diesem Modus ist die maximale Übertragungsrate von 20kBaud auf 10kBaud reduziert. Dies führt zu einer Verringerung der erzeugten EME. Der einzige Unterschied zwischen Normal Slope Mode und Low Slope Mode ist die Bus Signal Transition Zeit (Übergangszeit eines Signalwechsels). Diese ist im Low Slope Mode zwei mal grösser als im normal Slope Mode. In folgender Grafik wird dies gezeigt:



In den Low Slope Mode kann nur aus dem Sleep Mode oder aus dem Standby Mode gewechselt werden. Ein direkter Wechsel von Normal Slope Mode zu Low Slope ist nicht möglich.

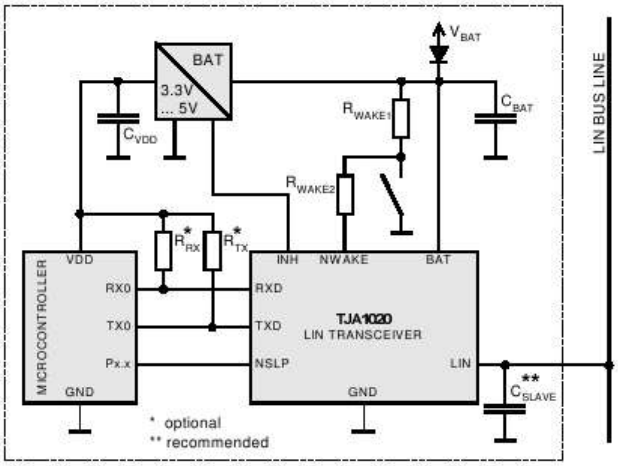
Der Low Slope Mode wird aktiveirt durch ein Low-Level an Pin TXD für die Zeit Tgotonom, min. in Verbindung mit einem High-Level an Pin NSLP für die Zeit Tgotonorm, max. Danach ist der Low Slope Mode ausgewält. Folgende Graphik verdeutlicht den Wechsel in den Low Slope Mode:



Slave Applikation

Ein schematischer Aufbau eines LIN-Tranceivers TJA1020 verbunden mit einem Mikrocontroller zeigt rechtsstehende Grafik. Der Protokoll Controller, bei uns der Infinion 515, ist über eine serielle Schnittstelle mit dem LIN-Transceiver verbunden. Hierbei ist der Pin TXD des TJA1020 der Daten Input des Bausteins. Der Pin RXD ist Empfangs-

Datenausgang. Der Schlafkontroll-Pin NSLP des TJA1020 wird direkt an den Mikrocontroller angeschlossen und somit von diesem gesteuert. Intern besitzt der TJA1020 noch einen Slave Terminierungs-Widerstand, der für die LIN-Kommunikation gebraucht wird.

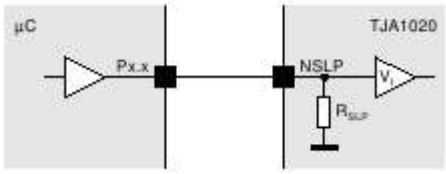


2.4.2 Verwendete Pins

NSLP Pin:

Der Pin NSLP ist das Schlafkontroll-Bit des TJA1020. Dieses unterstützt einen internen Pull-Down Widerstand Rslp. Der Pin NSLP wird benötigt um Betriebsmodi zu ändern.

Einfache NSLP Pin Anwendung:



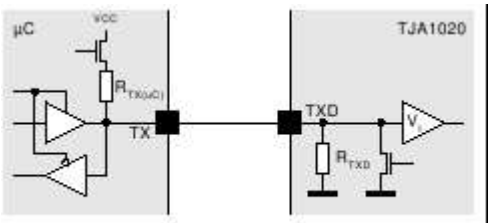
TXD Pin:

Der TXD Pin ist ein bidirektionaler Pin. Im Normal Slope und Low Slope Mode wird der Pin als normale Datensende-Eingangsleitung benutzt, im Standby Mode wird er jedoch als Wake-Up Signal quelle genutzt.

Will man einen lokalen Wake-Up mit Hilfe des Pins NWAKE durchführen, dann muss während der Aufwach-Phase der Pin TXD auf log. 1 angehoben werden. Es gibt zwei Möglichkeiten um das anheben auf log.1 zu realisieren.

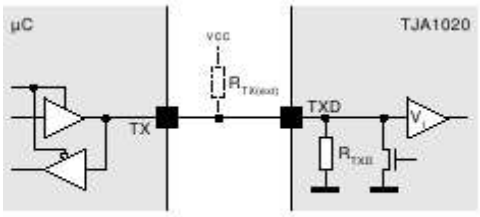
a.) Der Mikrocontroller Port Pin wird durch einen internen Pull-Up Widerstand unterstützt

oder



a) for µC with internal programmable pull-up

b.) ein externer Pull-Up Widerstand verbindet VCC und TXD.



b) for µC without internal pull-up

Wird kein lokaler Wake-Up benötigt, also Pin NWAKE wird nicht genutzt, dann wird der Pull-

Up Wiederstand Rtx nicht benötigt.

Wir haben keinen lokalen Wake-Up im Programm benutzt, haben aber trotzdem den Pin NWAKE auf der Platine fest verdrahtet, um uns diese Erweiterung noch offen zu halten.

Um einen lokalen Wake-Up durchzuführen, benötigt man einen Pull-Up Widerstand Rtx der mit Hilfe der folgenden Formeln bestimmt werden kann:

Min. high level pull-up current @ $V_{TXD(LC)} > V_{IH(TXD)min}$:

$$I_{HIGH\backslash RTX(),min} = \frac{V_{IH\backslash TXD(),min}}{R_{TXD,min}} + I_{IL\backslash TXD(),max}$$

Max. low level pull-up current @ $V_{TXD(LC)} < V_{IL(TXD)max}$:

$$I_{LOW\backslash RTX(),max} = \frac{V_{IL\backslash TXD(),max}}{V_{TXD}} I_{OL\backslash TXD(),min} \qquad \text{with } V_{TXD} = 0.4V$$

Range of pull-up resistor:

$$R_{TX,min} < R_{TX} < R_{TX,max} \qquad \text{with}$$

$$R_{TX,min} = \frac{VCC_{max} - V_{IL\backslash TXD(),max}}{I_{LOW\backslash RTX(),max}} \text{ and } R_{TX,max} = \frac{VCC_{min} - V_{IH\backslash TXD(),min}}{I_{HIGH\backslash RTX(),min}}$$

with [1]

$V_{IH(TXD)min}$	minimum TXD HIGH-level input voltage
$V_{IL(TXD)max}$	maximum TXD LOW-level input voltage
R_{TXDmin}	minimum TXD pull-down resistor
$I_{IL(TXD)max}$	maximum TXD LOW-level input current
$I_{OL(TXD)min}$	minimum TXD LOW-level output current

Die Versorgungsspannung VCC des Mikrocontrollers sei hierbei 5V, dann können Pull-Up Widerstände zwischen den hier errechneten maximal-/minimal-Werten gewählt werden:

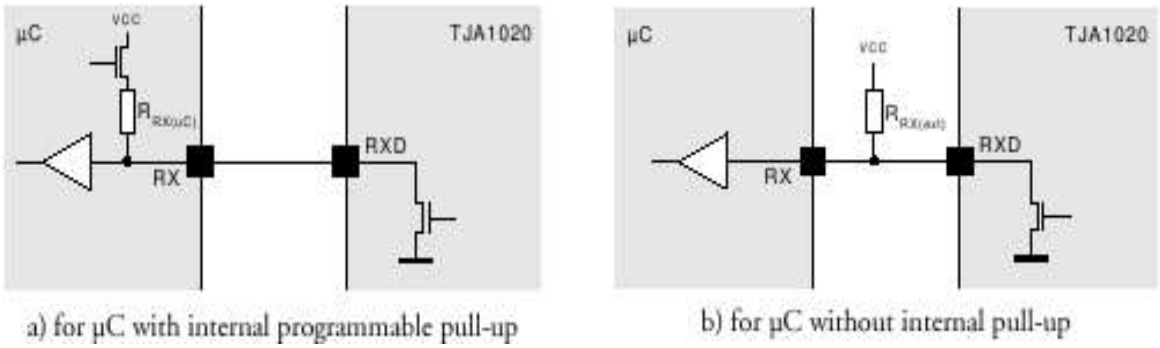
$$R_{TX,min} = \frac{VCC_{max} - V_{IL\backslash TXD(),max}}{I_{LOW\backslash RTX(),max}} = 1.4k\Omega \quad \text{with } I_{LOW\backslash RTX(),max} = \frac{V_{IL\backslash TXD(),max}}{V_{TXD}} I_{OL\backslash TXD(),min} = 3mA$$

$$R_{TX,max} = \frac{VCC_{min} - V_{IH\backslash TXD(),min}}{I_{HIGH\backslash RTX(),min}} \approx 140k\Omega \quad \text{with } I_{HIGH\backslash RTX(),min} = \frac{V_{IH\backslash TXD(),min}}{R_{TXD,min}} + I_{IL\backslash TXD(),max} = 21\mu A$$

Ein empfohlener Wert für die Pull-Up Widerstände Rtx ist 2,2 KOhm.

RXD Pin:

Der Empfangsdaten-Ausgang RXD kann von 3,3V Mikrocontroller Systemen so wie auch von 5V Systemen verwendet werden, wenn der auf der Mikrocontroller Platine verwendete Pin einen internen Pull-Up Widerstand besitzt. Sollte dies nicht der Fall sein, dann ist ein externer Pull-Up Widerstand zwischen VCC und RXD anzubringen. Folgendes Schaubild wird dies verdeutlichen:



a) for µC with internal programmable pull-up

b) for µC without internal pull-up

Folgende Formeln können benutzt werden um die Größen der Pull-Up Widerstände zu berechnen:

$$R_{RX(min)} < R_{RX} < R_{RX(max)} \quad \text{with}$$

$$R_{RX(min)} = \frac{VCC_{max} - V_{L\ (RX)(max)}}{V_{L\ (RX)(max)}} \cdot \frac{V_{RXD}}{I_{L\ (RXD)(min)}}, \qquad V_{RXD} = 0.4V \quad \text{and}$$

$$R_{RX(max)} = \frac{VCC_{min} - V_{HIGH(RX)(min)}}{I_{LH(RXD)(max)}}$$

with

$I_{LH(RXD)(max)}$	maximum RXD HIGH-level leakage current [1]
$I_{OL(RXD)(min)}$	minimum RXD LOW-level output current [1]
$V_{HIGH(RX)(min)}$	minimum µC port pin (RX) HIGH-level input voltage
$V_{LOW(RX)(max)}$	maximum µC port pin (RX) LOW-level input voltage

Ein guter Wert für den Widerstand Rrx liegt ebenfalls bei 2,2KOhm.

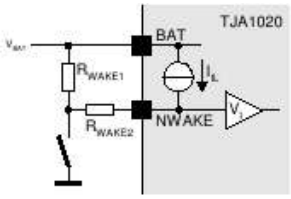
NWAKE Pin:

Der lokale Eingangspin NWAKE wird dazu benutzt ein lokales Wake-Up Event, welches durch eine fallende Flanke signalisiert wird, zu erkennen. Diese fallende Flanke wird durch ein halten des Low-Level Zustands für die Zeit Tnwake genutzt, um eine EMI Filterung zu garantieren.

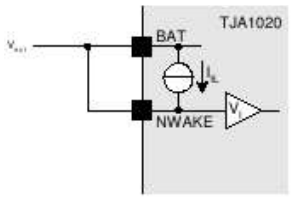
Wird Pin NWAKE nicht gebraucht dann kann er auf VCC gelegt werden, somit ist eine lokales Wake-Up via NWAKE aber nicht mehr möglich.

Folgende Grafiken zeigen wie NWAKE benutzt werden kann:

a.) Ein lokales Wake-Up mit externem Schalter

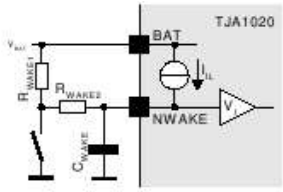


b.) Eine Applikation die NWAKE nicht benutzt

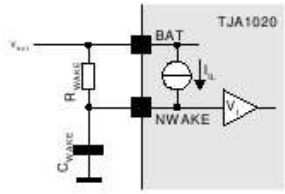


Sollte ein automatisches Aufwachen des TJA1020 nach dem Anschalten gewünscht sein, so kann dies durch eine RC-Kombination an Pin NWAKE gelöst werden. Folgende Grafiken stellen dies dar:

a.) Lokaler Wake-Up mit externem Schalter oder automatischer Wake-Up



b.) Für Applikation ohne lokalen Wake-Up

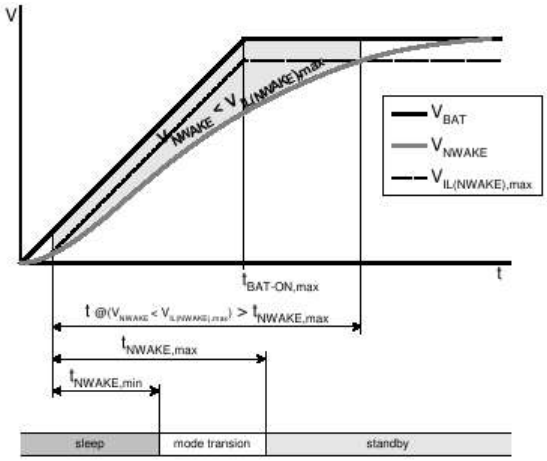


Für die RC-Kombination gilt folgende Regel:

Der Pull-Up Widerstand Rwake sollte 10KOhm sein, dann errechnet sich der Kondensator Cwake wie folgt:

$$C_{WAKE} = \frac{t_{BAT\text{-}ON,max}}{R_{WAKE}} = 100nF \quad \text{with} \quad t_{BAT\text{-}ON,max} = 1ms \; > \; 2t_{NWAKE,max} = 100\mu s$$

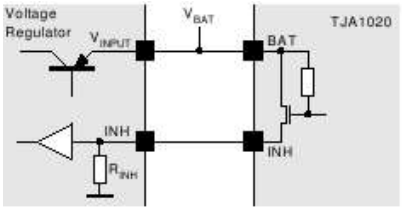
Rechtsstehende Grafik zeigt das Timing nach dem einschalten des TJA1020:



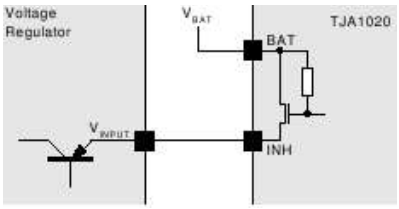
INH Pin:

Der Pin INH ist ein Batterie-abhängiger Open-Drain Ausgang mit dem ein externer Spannungsregler kontrolliert werden kann. Für diesen Zweck wird ein externer Pull-Down Widerstand Rinh, der an Masse angeschlossen wird, benötigt. Dieser Pull-Down Wiederstand ist normalerweise im Spannungsregler selbst schon integriert - muss also nicht mehr angebracht werden. In den nächsten Grafiken wird eine typische Benutzung des Pins INH gezeigt:

a.) Spannungsregler mit sperrendem Eingang



b.) Spannungsregler ohne sperrenden Eingang



LIN Pin:

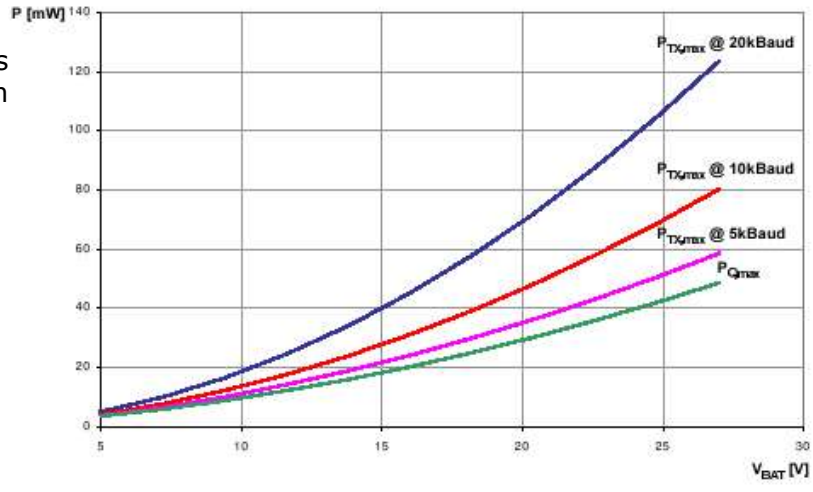
Der Pin LIN wird dafür benutzt, um Daten vom LIN-Bus zu empfangen oder Daten über ihn zu verschicken.

Verwendete Abkürzungen:

SYMBOL	PARAMETER	CONDITIONS	MIN.	TYP.	MAX.	UNIT
$\Delta t_{slope(norm)}$	normal slope symmetry	normal slope mode; C_L = 10 nF; R_L = 500 Ω; V_{BAT} = 12 V; $t_{f(slope)(dom)} - t_{r(slope)(rec)}$	−5	0	+5	µs
$t_{f(slope)(norm)(dom)}$	normal slope fall time LIN (100% to 0%)	normal slope mode; C_L = 6.8 nF; R_L = 660 Ω; V_{BAT} = 12 V; transition from recessive to dominant; note 4	–	12	22.5	µs
$t_{r(slope)(norm)(rec)}$	normal slope rise time LIN (0% to 100%)	normal slope mode; C_L = 6.8 nF; R_L = 660 Ω; V_{BAT} = 12 V; transition from dominant to recessive; note 5	–	12	22.5	µs
$\Delta t_{slope(norm)}$	normal slope symmetry	normal slope mode; C_L = 6.8 nF; R_L = 660 Ω; V_{BAT} = 12 V; $t_{f(slope)(dom)} - t_{r(slope)(rec)}$	−4	0	+4	µs
$t_{f(slope)(low)(dom)}$	low slope fall time LIN (100% to 0%)	low slope mode; C_L = 10 nF; R_L = 500 Ω; V_{BAT} = 12 V; note 4	–	30	62	µs
$t_{r(slope)(low)(rec)}$	low slope rise time LIN (0% to 100%)	low slope mode; C_L = 10 nF; R_L = 500 Ω; V_{BAT} = 12 V; note 5	–	30	62	µs
t_{BUS}	dominant time for wake-up via bus	sleep mode	30	70	150	µs
t_{NWAKE}	dominant time for wake-up via pin NWAKE	sleep mode	7	20	50	µs
$t_{gotonorm}$	time period for mode change from sleep or standby mode into normal/low slope mode		2	5	10	µs
$t_{gotosleep}$	time period for mode change from normal/low slope mode into sleep mode		2	5	10	µs
t_{dom}	TXD dominant time out	V_{TXD} = 0 V	6	12	20	ms

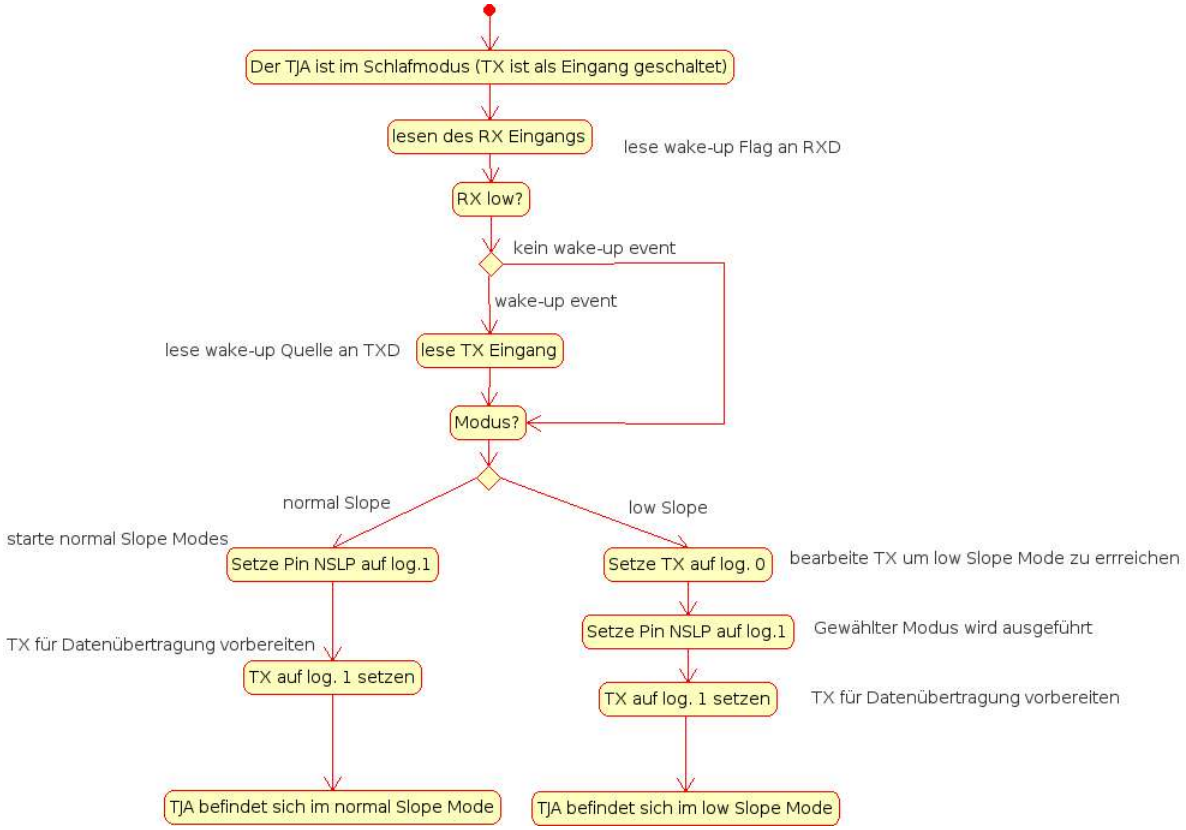
Leistungsaufnahme:

Die Leistungsaufnahme des TJA1020 variiert je nach benutzter Baud Rate.

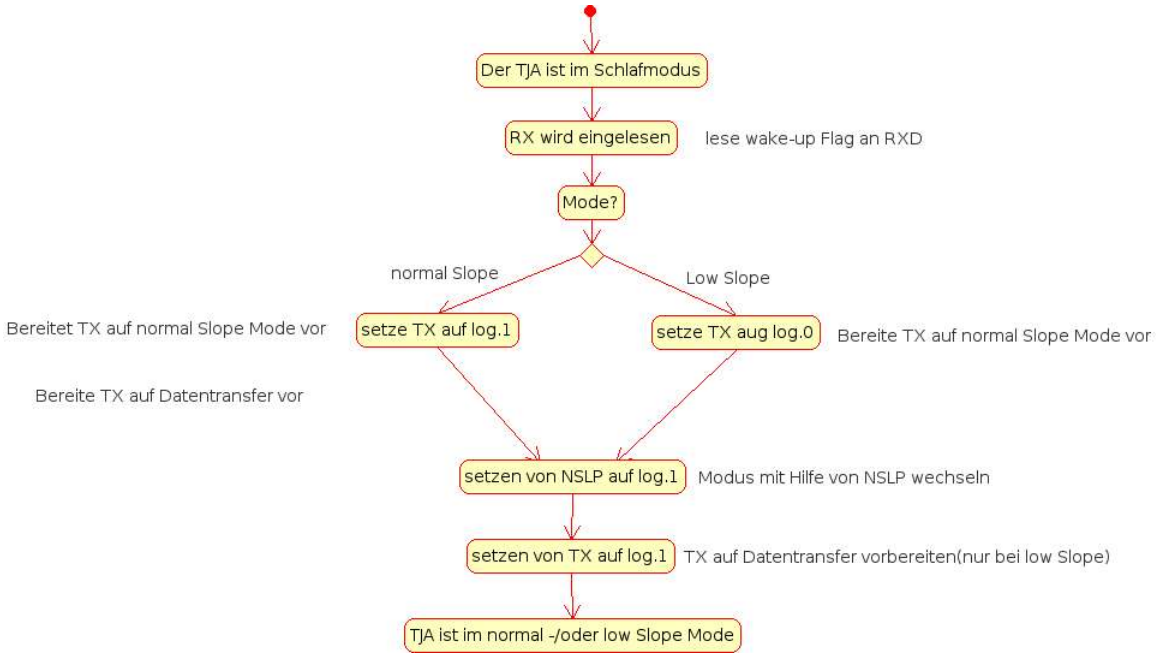


Applikationsdiagramme:

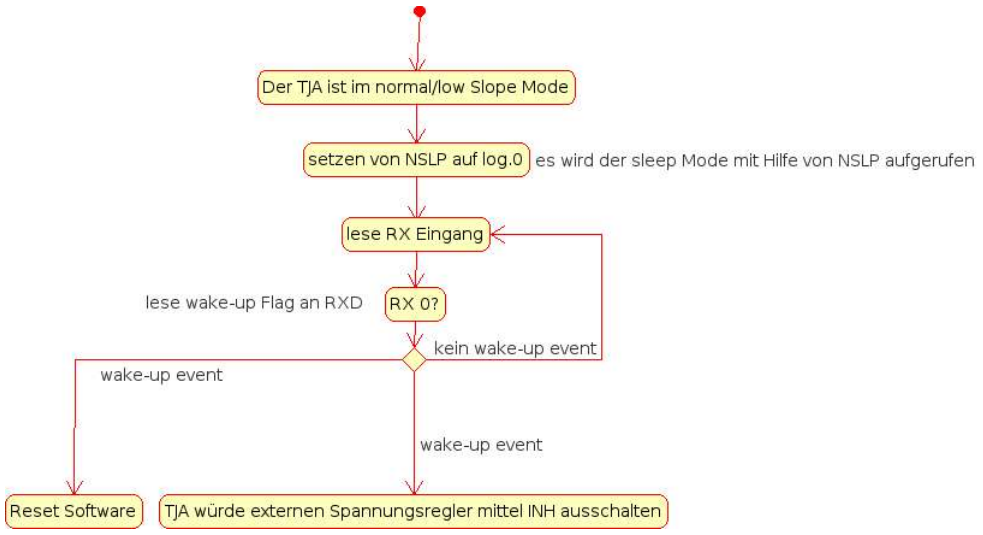
Anweisungs-Diagramm wie vom Sleep zum Normal Slope Modus gewechselt werden kann, unter Verwendung von NWAKE. Dieser Ablaufplan wurde für einen nicht in den Sleep Modus fallenden Mikrocontroller erstellt:



Im folgenden Ablaufplan wird gezeigt, wie ohne den Pin NWAKE in den Normal- oder Low Slope Mode gewechselt werden kann.

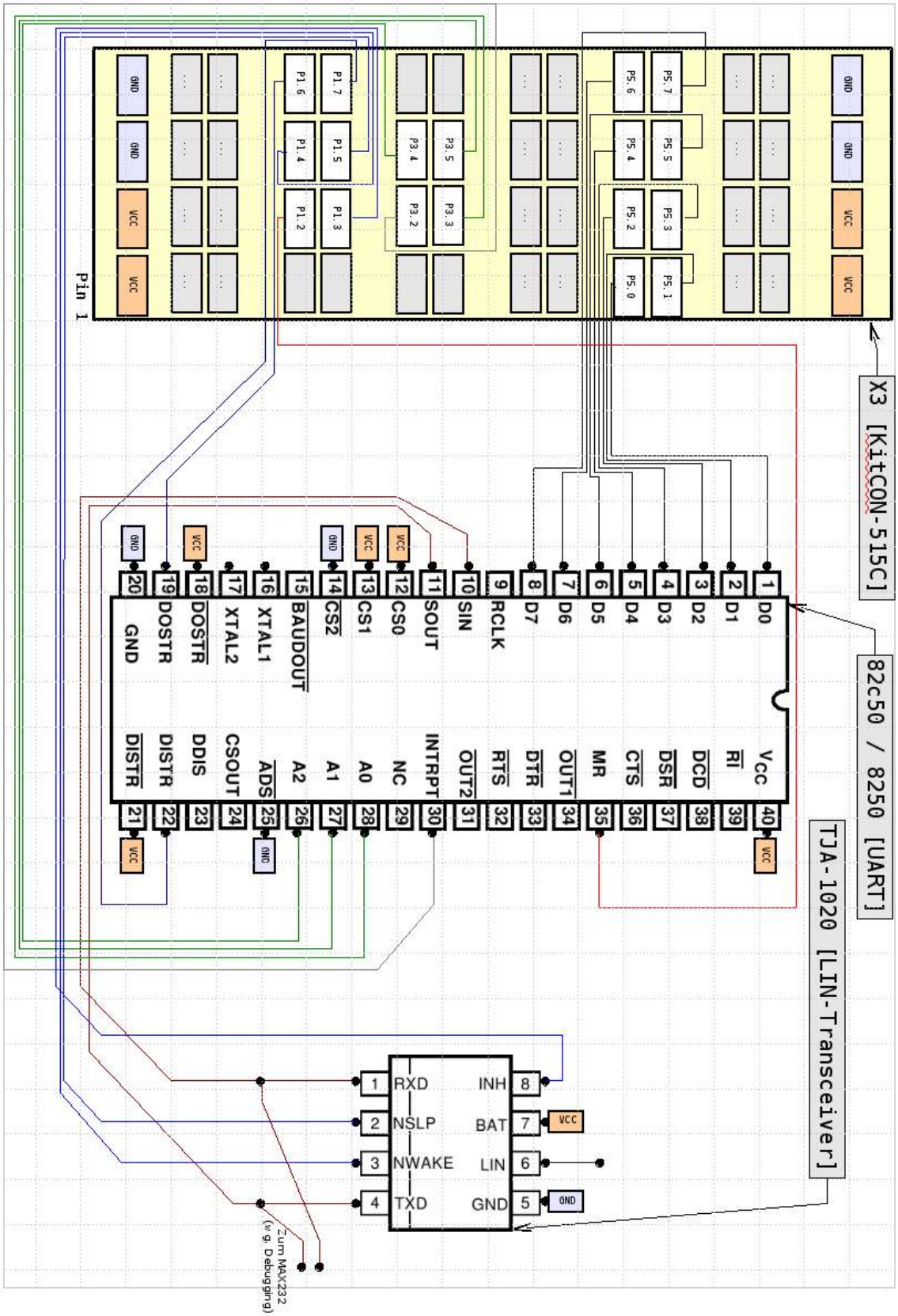


Zu guter Letzt wird noch ein Ablaufplan gezeigt der den Vorgang beschreibt, wie man vom Normal-/Low Slope Mode zum Sleep Mode gelangt, ohne Verwendung von Pin NWAKE.



2.5 Verdrahtungsplan

Im folgenden der Verdrahtungsplan der gesamten Schaltung:



3 Software

3.1 Entwicklungsumgebung KEIL µVision2

Zur Programmierung des Microcontrollers haben wir die Integrierte Entwicklungsumgebung "µvision2" der Firma Keil in der Version 2.33 verwendet. Der erzeugt Maschinencode wird hierbei direkt von der Entwicklungsumgebung per serieller Schnittstelle in den Flash-Speicher des Microcontrollers übertragen.

3.2 Hardware Low-Level Funktionen

Alle Low-Level Funktionen hier im Überblick:

Modul / Datei	Funktionen / Aufgabe	
C515-IO	void **setLEDs**(char wert)	Schaltet onboard-LEDs an/aus
	char **getSwitches**()	Gibt Schalterstellungen zurück
UART-8250	void **initUART**()	Initialisiert Baustein (Reset, ...)
	void **confUART**()	Konfiguration (Baudrate, ...)
	void **setAdr**(char, char , char)	Hilfsfunktions zum Anlegen einer Adresse
	char **read**()	Hilfsfunktions zum lesen von Daten
	void **write**(char)	Hilfsfunktions zum schreiben von Daten
	void **writeTHR**(char)	Beschreibt THR des UART
	void **writeLCR**(char)	Beschreibt LCR des UART
	void **writeLSR**(char)	Beschreibt LSR des UART
	void **writeMSR**(char)	Beschreibt MSR des UART
	char **readRBR**()	Gibt Inhalt des RBR vom UART zurück
	char **readLCR**()	Gibt Inhalt des LCR vom UART zurück
	char **readLSR**()	Gibt Inhalt des LSR vom UART zurück
	char **readIIR**()	Gibt Inhalt des IIR vom UART zurück
TJA-1020	void **initTJA**()	Initialisiert Baustein (NWAKE=1)
	void **startNSM**()	Versetzt den TJA in den Normal Slope Mode
	void **startLSM**()	Versetzt den TJA in den LowSlope Mode
	void **startSLM**()	Versetzt den TJA in den Sleep Mode
	void **startSBM**()	Versetzt den TJA in den Standby Mode

3.3 LIN Low-Level Funktionen

Die angedachten[2] grundlegenden LIN-Funktionen befinden sich in einem eigenen Modul "LINframe" - hier ein Überblick:

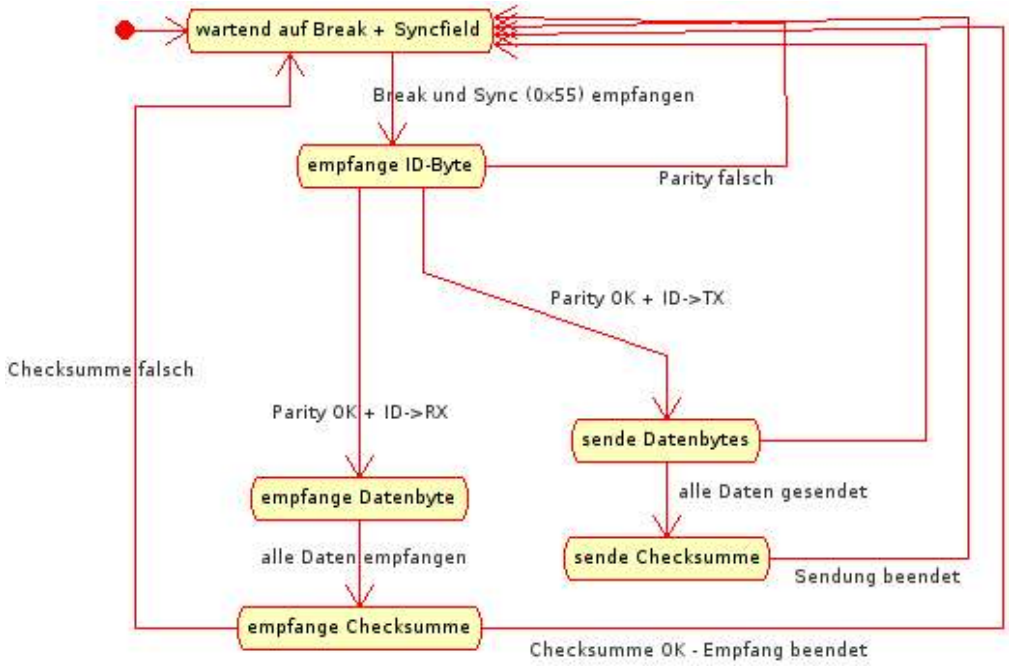
LINframe - Funktionen	
char **LIN_getID**(LINframe*)	Gibt ID des LIN-Frames direkt als Wert zurück
char **LIN_getLength**(LINframe*)	Gibt Länge des Frames direkt als Wert zurück
Bool **LIN_parityOK**(LINframe*)	Prüft, ob Parität des ID-Bytes OK ist
char **LIN_calcChksum**(LINframe*)	Berechnet die Checksumme der Daten-Bytes
Bool **LIN_chksumOK**(LINframe*)	Prüft, ob die Checksumme der Daten-Bytes OK ist

LINframe* ist ein Zeiger auf folgende in LINframe.h definierte Struktur:

char ID
char daten[8]
char chksum

3.4 LIN-Slave

Hier ein Überblick über den Ablauf im Slave-Task:



2 sh. Abschnitt Probleme – leider konnten diese Funktionen bisher nicht zum Einsatz kommen und sind teilweise auch noch nicht implementiert.

Der LIN-Slave wartet zuerst auf einen Sync-Break und dann auf den Sync-Frame mit dem Wert 0x55.

Anschliessend wird der Identifier empfangen und ausgewertet:

- Wenn Parität nicht OK – Abbruch des Frame-Empfangs
- Wenn ID als Versand-ID bestimmt wurde – Daten und generierte Checksumme senden
- Wenn ID als Empfangs-ID bestimmt wurde – Daten annehmen und Checksumme prüfen.

Ein evtl. empfangener Break sollte an jeder Stelle erkannt werden und zum Abbruch des gerade verarbeitenden Frames führen. Das Ganze läuft in einer Endlosschleife, so dass gleich der nächste Frame verarbeitet werden kann.

4 Probleme

Während der Entwicklungsphase sind wir auf mehrere Probleme gestoßen, die sich aber fast alle als lösbar herausstellten.

Zu den lösbaren Problemen gehört unter anderem auch die (möglicherweise) falsch eingestellte Baudrate des Bausteins UART 82c50. Aber mit einer kleinen Änderung in der Initialisierung ist dies aber behoben.

Mit etwas höherem Aufwand ist das Anbringen der Transistor-Widerstände verbunden. In unserem Aufbau fehlten die passenden Transistor-Widerstände die dem TJA1020 LIN-Transceiver vorgeschaltet werden müssen. Sie müssen einfach nur nachträglich in die Schaltung eingelötet werden.

Anders sieht es mit dem Defekt des UART 82c50 Bausteins aus, welcher uns dazu zwang unsere Projektarbeit frühzeitig zu beenden. Der Defekt wirkt sich dahingehend aus, dass das Sendepufferregisterflag nicht richtig gesetzt wird, d.h. es ist immer HIGH (Log 1) auch wenn der Puffer leer ist. Solange dieses Flag sich nicht auf den LOW-Zustand (Log 0) geht, nimmt der Baustein kein weiteres Zeichen zum Senden auf, somit war es uns unmöglich ein Zeichen zu senden.

Um Programmierfehler auszuschließen wurde der UART 82c50 über den Microcontroller, durch ansteuern des Masterresets, resettet. Nach einem Reset hat der Baustein definierte Werte in den Registern, die sich auch verändern und wieder auslesen ließen – allerdings begann der Chip kurz nach dem Reset sinnlos wirre Zeichen zu senden.

Da wir keinen funktionsfähigen LIN-Slave gebaut hatten, ließen wir auch die entspr. Windows-Applikation weg, da sie nicht getestet werden konnte.

Trotz des nicht funktionierenden LIN-Slaves wurden alle benötigten Low-Level-Funktionen ausprogrammiert und in das Programm des Mikrocontrollers integriert. Für die funktionalität war es unter anderem auch nötig ein LinFrame im Programm abzubilden, entsprechend wurden auch Funktionen geschrieben die mit der LinFrame-Struktur richtig umgehen konnte.

Nebem dem theoretischen und programmiertechnischem Teil wurde auch eine Schaltung auf dem Phytec Entwicklerboard KitCON-515C aufgebaut und weitestgehend getestet. Leider waren diese Tests nicht sehr ergiebig wegen dem oben geschildertem Hardwaredefekt.

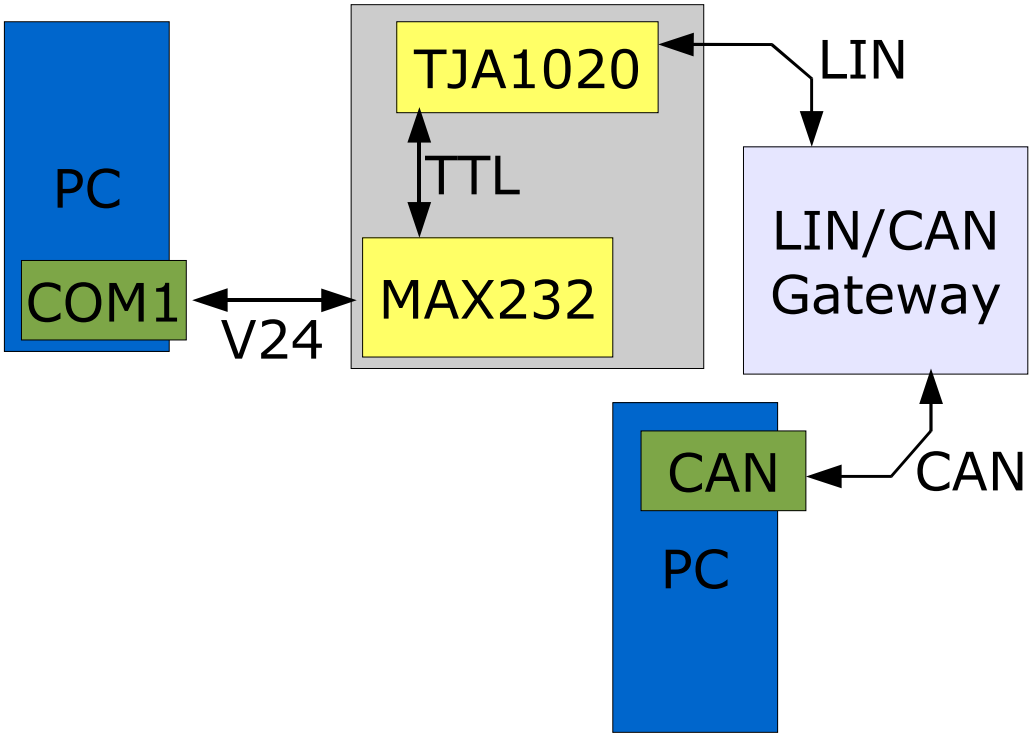
Nach Erkennen des Defekts der seriellen Schnittstelle, wurde ein alternativer Aufbau erstellt, der anstelle eines Mikrokontrollers einen PC mit seriellem Anschluss verwendet. Aufgrund des Zeitmangels wurde dieser Aufbau nicht hinreichend getestet.

5 Alternativ-Implementierung

Da wir durch oben genannte Probleme mit dem vermutlich defekten UART erst gar nicht dazu kamen, uns auch praktisch mit dem LIN-Bus zu beschäftigen, entstand relativ kurz vor dem Abgabetermin der Projektarbeit folgende Idee für eine einfachere Alternative:

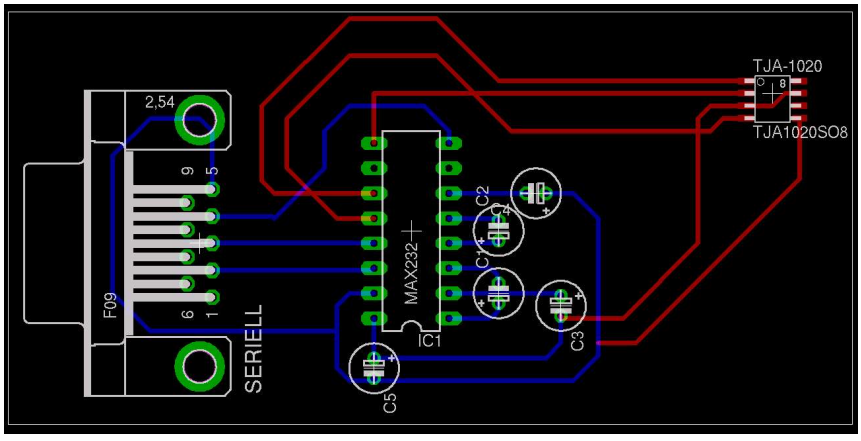
Warum nicht die serielle Schnittstelle eines herkömmlichen PCs dazu verwenden, um direkt über einen Pegelwandler mit dem LIN-Transceiver – und damit über den LIN-Bus - zu kommunizieren?

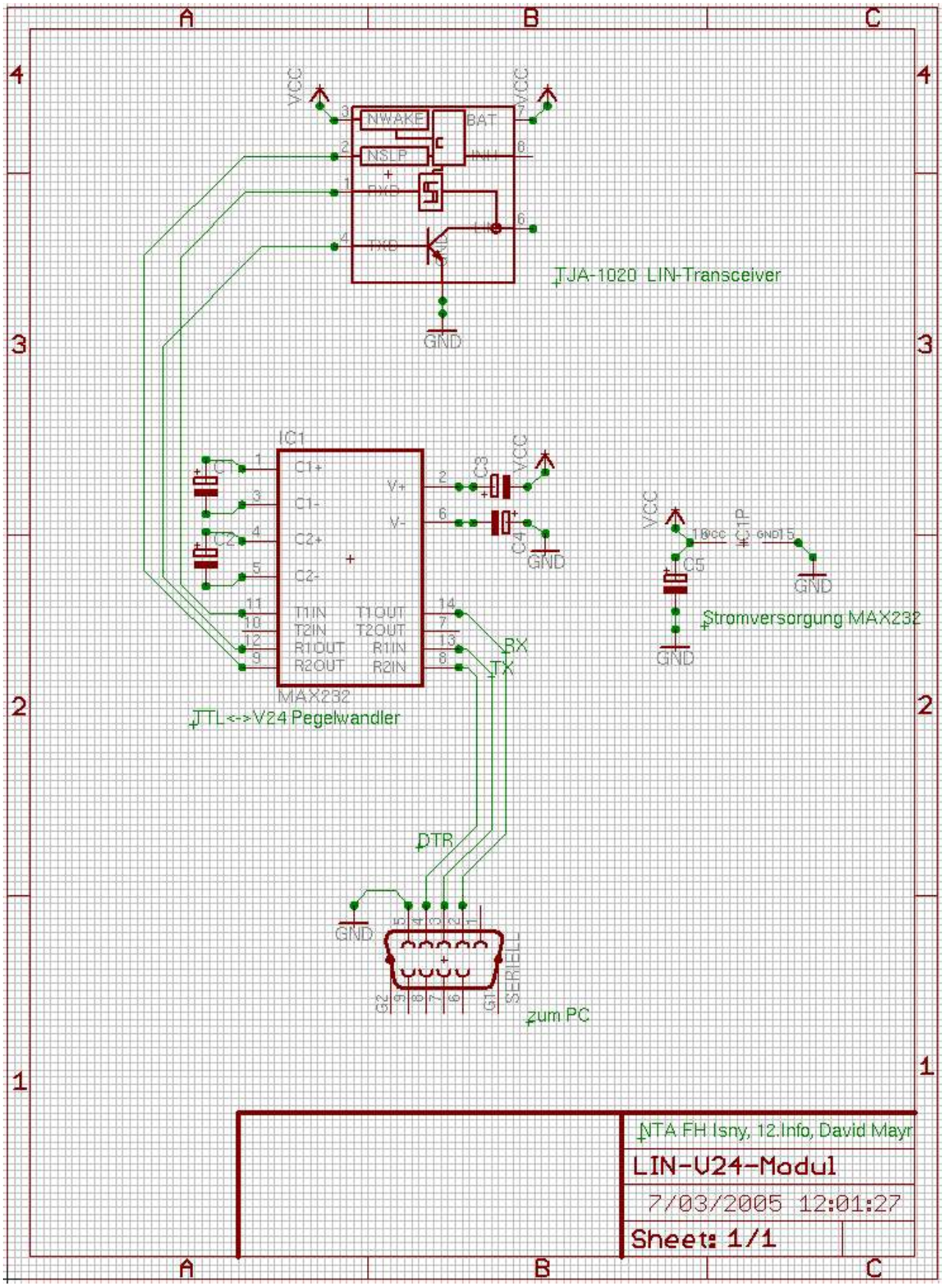
Der Gesamt-Aufbau könnte folgendermaßen aussehen:



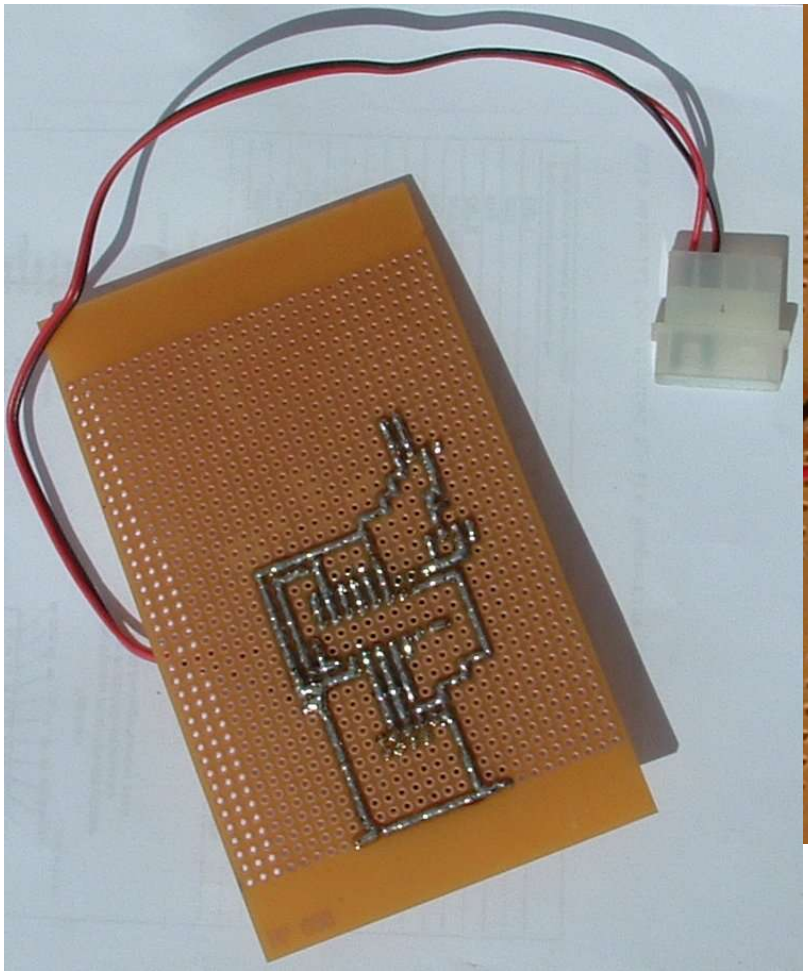
Damit ließe sich ein recht flexibler – wenn auch überdimensionierter – LIN-Slave oder evtl. auch LIN-Master in Form eines PCs implementieren.

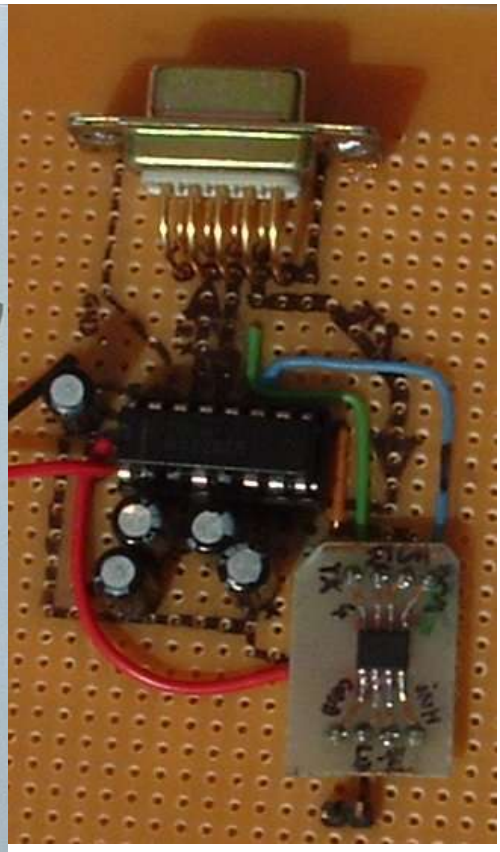
Hier der Schaltplan, das Platinen-Layout und die fertige Platine:




A
B
C
1
2
3
4
VCC
NWAKE
NSLP
RXD
TXD
BAT
INH
LIN
GND
TJA-1020 LIN-Transceiver
IC1
C1+
C1-
C2+
C2-
V+
V-
T1IN
T2IN
R1OUT
R2OUT
T1OUT
T2OUT
R1IN
R2IN
MAX232
TTL<->V24 Pegelwandler
C1
C2
C3
C4
C5
Stromversorgung MAX232
RX
TX
DTR
SERIELL
zum PC
NTA FH Isny, 12.Info, David Mayr
LIN-U24-Modul
7/03/2005 12:01:27
Sheet: 1/1

Die Schaltung wurde direkt auf eine Lochraster-Platine gelötet. Die nötige Versorgungsspannung von 5V (ca. 15mA) wurde direkt vom PC über einen Festplatten-Stromstecker abgegriffen.





Bisher wurde ein noch nicht vollständig funktionierender LIN-Slave in C für das freie Betriebssystem LINUX implementiert, dessen Funktionen sich an denen der Implementierung für den C515-Mikrocontroller anlehnen.

Im folgenden sind die Quellcode-Dateien und deren Funktionen im Überblick kurz beschrieben:

UART.c	
void **UART_init**()	Initialisierung des UART – IO-Berechtigungen freigeben
void **UART_conf**()	Konfiguration des UART – Baudrate=9600, 8 Datenbit, keine Parität, 1 Stoppbit, ...
void **UART_setDTR**(bool)	DTR-Leitung direkt beeinflussen – ist indirekt verbunden mit NSLP-Pin des TJA1020
void **UART_setBreak**(bool)	Break auf TX-Leitung an / aus (TX=Dauer-LOW, senden deaktiviert)
bool **UART_checkBreak**()	Prüfen, ob Break auf TX-Leitung erkannt / empfangen wurde
void **UART_send**(char)	Senden eines Bytes, sobald UART bereit (THR leer)
char **UART_receive**()	Empfangen und zurückgeben eines Bytes

In der dazugehörigen Headerdatei UART.h wurde neben den Funktionsprototypen auch die Basis-IO-Adresse und die Registernamen (THR, RBR, ...) des UART hinterlegt.

TJA1020.c	
void **TJA_send**(char)	Senden eines Bytes – verwendet UART_send()
char **TJA_receive**()	Senden eines Bytes – verwendet UART_receive()
void **TJA_setNSLP**(bool)	Ändere NSLP-Pin – um Slope-Mode zu ändern. Verwendet UART_setDTR()
void **TJA_disableTX**(bool)	Lege TX auf LOW um Sync-Break erzeugen zu können. Verwendet UART_setBreak()
void **TJA_sendSyncBreak**()	Erzeugt Sync-Break mit TJA_disableTX() für 768 µs
bool **TJA_checkSyncBreak**()	Prüft, ob Sync-Break erkannt / empfangen wurde - verwendet UART_checkBreak()
bool **TJA_waitSyncBreak**()	Wartet, bis Sync-Break erkannt wurde
void **TJA_waitNoSyncBreak**()	Wartet, bis das Ende des Sync-Break erkannt wurde
void **TJA_gotoNSMode**()	Versetzt den TJA1020 in den Normal Slope Mode
void **TJA_gotoLSMode**()	Versetzt den TJA1020 in den Low Slope Mode

LINframe.c	
char **LIN_getID**(LINframe*)	Gibt ID des LIN-Frames direkt als Wert zurück
char **LIN_getLength**(LINframe*)	Gibt Länge des Frames direkt als Wert zurück
Bool **LIN_parityOK**(LINframe*)	Prüft, ob Parität des ID-Bytes OK ist (verwendet LIN_calcChksum())
char **LIN_calcChksum**(LINframe*)	Berechnet die Checksumme der Daten-Bytes
Bool **LIN_chksumOK**(LINframe*)	Prüft, ob die Checksumme der Daten-Bytes OK ist (verwendet LIN_calcChksum())

In der dazugehörigen Headerdatei LINframe.h wurde den Funktionsprototypen auch die Struktur eines Frames als neuer Datentyp mit dem Namen "LINframe" definiert.

LINslave.c	
void **LIN_startSlave**( )	Startet die Endlosschleife der LINslave-Routine

Die main()-Funktion steckt in einer eigenen Datei mit dem Namen "SLAVEtest.c" und gestaltet sich folgendermaßen:

```
void main()
{
        UART_init();          // UART initialisieren
        UART_conf();          // und konfigurieren

        TJA_gotoNSMode();     // LIN-Transceiver aktivieren (NormalSlope Mode)

        LIN_startSlave();     // Starte LIN Slave (Endlosschleife)
}
```

Da die Zeit recht knapp war, wurde dieser alternative Ansatz bisher noch nicht vollständig realisiert und getestet. Bisher ist es allerdings schon möglich, die vom LIN/CAN-Gateway ausgesendeten Sync-Break / Sync-Field – Folgen richtig zu erkennen.

6 Fazit

Wir möchten uns bei SiE Sontheim Industrieelektronik GmbH bedanken, welche uns freundlicherweise eine LIN-CAN-Gateway für dieses Projekt zur Verfügung gestellt hatte - auch wenn es bisher zu unserem Bedauern noch nicht ausgiebig verwendet wurde.

Unser Team hat Aufgrund seiner guten Zusammenarbeit und Kommunikationsfreudigkeit viel Spass an dieser Projektarbeit gehabt. Auftretende "Nebenprobleme" wurden meist in kurzer Zeit behoben und somit konnte viel an dem eigentlichen Problem, dem Lin-Slave, gearbeitet werden. Vorallem die Theorie und das Programm konnten schnell erarbeitet und erstellt werden. Im Gegensatz dazu stand die Hardware und der Aufbau, was uns immer wieder vor neue Herausvorderungen stellte. Manche dieser Probleme waren uns unerklärlich und wären wohl ohne die Hilfe von Herrn Gerum, dem wir an dieser Stelle herzlich danken möchten, nicht gelöst worden.

Diese Projektarbeit lässt sich am besten mit einem Umgangsprachlichem Motto beschreiben: "Leichter gesagt als getan!"

7 Quellenangaben

Scripte

Microprozessor (B. Gerum)

Bussysteme und Interfaces (B. Gerum)

Datenblätter

82c50
MAX232
Philips TJA1020
Phytec kitCon-515C

LIN-Spezifikation von lin-subbus.org

Internet

http://www.google.de
http://www.elektroniknet.de/
http://www.lin-subbus.org/

8 Anhang

8.1 Quellcode – Mikrocontroller

8.1.1 EXTRAs.c

```
/*
 *    Datei mit global verwendetten Extra-Funktionen
 *    ----------------------------------------------
 *    2005.06
 *    Matthias Müller,  David Mayr,  Martin Junginger
 */


/*
 *   wait: einfache Warteschleife
 *   ========
 */
void wait( char wert )
{
    int i;
    for( i=0; i<=wert; i++ );
}
```

8.1.2 C515-IO.c

```
/*
 *    IO-Funktionen des C515C Microcontroller (LEDs/Schalter)
 *    -------------------------------------
 *    2005.06
 *    Matthias Müller,  David Mayr,  Martin Junginger
 */

#include "REG515.h"

/*
 *   setLEDs: setzt die fest verdrahteten LEDs a.d. kitCON-Board an Port 4
 *   =======
 */
void setLEDs( char wert )
{
    P4 = !wert; // Port 4 setzen (LEDs) - Achtung: invertieren
}


/*
 *   getSwitches: liest die Schalterstellung(en) ein
 *   ===========
 */
char getSwitches()
{
    return 255; // >> noch nicht implementiert
}
```

8.1.3 UART-8250.c

```
/*
 *    Treiberdatei für UART Baustein 8250
 *    -----------------------------------
 *    2005.06
 *    Matthias Müller,  David Mayr,  Martin Junginger
 */

#include "UART-8250.h"       // Asynchrone Schnittstelle einbinden
#include "REG515.H"          // Port- & Pin-Namen einbinden

/*
 *   initUART: Initialisierungsfunktion des UART
 *   ========
 */
void initUART()
{
    MR = 1;      // Master-Reset einmal mit pos. Flanke auslösen
    MR = 0;      // und dann zurücksetzen
    DISTR = 0;   // Data In  Strobe deaktivieren
    DOSTR = 0;   // Data Out Strobe deaktivieren
}

/*
 *   confUART: Konfigurationsfunktion des UART (Baudrate, LCR, ...)
 *   ========
 */
void confUART()
{
    writeLSB( 0x12 );        // LSB des Baudratendivisors einstellen: bei 9600Baud 0x0C; bei 19200Baud 0x06
    writeMSB( 0x00 );        // MSB des Baudratendivisors einstellen: bei 9600Baud 0x00; bei 19200Baud 0x00
    writeLCR( 0x03 );        // im LCR Schnittstelle konfigurieren: 8 Datenbits, 1 Stopbit, Parität aus
}

/*
 *   setAdr: setzt eine übergebene Adresse an die Adressleitungen des UART
 *   ======
 */
void setAdr( char a2, char a1, char a0 )
{
    ADR0 = a0;
    ADR1 = a1;
    ADR2 = a2;
}

/*
 *   write: übergibt einen Wert an die Datenleitungen des UART
 *   =====
 */
```

```
void write( char wert )
{
    DOSTR = 1;          // Schreibvorgang auf UART starten
    P5 = wert;          // Daten auf Port 5 ausgeben
    DOSTR = 0;          // Schreibvorgang beenden
}


/*
 *   read: liest einen Wert von der Datenleitungen des UART
 *   ====
 */
char read()
{
    char wert;
    P5 = 0xFF;          // Pins des Port 5 als Eingang konfigurieren
    DISTR = 1;          // Lesevorgang von UART starten
    wert = P5;      // Daten von Port 5 in den temp. Puffer einlesen
    DISTR = 0;          // Lesevorgang beenden
    return wert;
}


/*
 *   readRBR [Receive Buffer Reg.]: liest ein Zeichen aus dem Empfangs-Register
 *   =======
 */
char readRBR()
{
    setAdr( 0,0,0 );    // RBR auswählen
    return read();
}


/*
 *   writeTHR [Transmit Holding Reg.]: schreibt ein Zeichen in das Versende-Register
 *   ========
 */
void writeTHR( char wert )
{
    setAdr( 0,0,0 );    // THR auswählen
    write( wert );
}


/*
 *   readLCR [Line Control Reg.]: liest das LCR-Register ein
 *   =======
 */
char readLCR()
{
    setAdr( 0,1,1 );    // LineControllRegister auswählen
    return read();
}
```

```
/*
 *   writeLCR [Line Control Reg.]: beschreibt das LCR-Register
 *   ========
 */
void writeLCR( char wert )
{
    setAdr( 0,1,1 );    // LineControllRegister auswählen
    write( wert );
}


/*
 *   readLSR [Line Status Reg.]: liest das LSR-Register ein
 *   =======
 */
char readLSR()
{
    setAdr( 1,0,1 );    // LSR auswählen
    return read();
}


/*
 *   readIIR [Interrupt Identification Reg.]: liest das IIR-Register ein
 *   =======
 */
char readIIR()
{
    char wert;

    setAdr( 0,1,0 );    // IIR auswählen
    wert = read();      // Wert einlesen

    if (wert & 1)       // Falls kein Interrupt vorliegt (Bit0=1),
        return 0xFF;    // Fehler mit Rückgabe von 255 signalisieren
    else                // sonst
        return wert;    // Interrupt zurückgeben
}


/*
 *   intEnable: aktiviert den Interrupt des UART
 *   =========
 */
void intEnable()
{
    setAdr( 0,0,1 );// IIR auswählen
    write( 0x03 );  // Interrupts auswählen (Received Data avail. & THR empty)
}


/*
 *   writeLSB [LSB f. Baudrate]
 *   ========
 */
```

```
void writeLSB( char wert )
{
    writeLCR( readLCR() | 0x80 );   // DLAB-Bit im LCR setzen und Rest erhalten

    setAdr( 0,0,0 );     // Divisor-Latch LSB auswählen
    write( wert );       // Wert auf UART ausgeben

    writeLCR( readLCR() & 0x7F );   // DLAB-Bit wieder aus LCR Register entfernen
}


/*
 *   writeMSB [MSB f. Baudrate]
 *   ========
 */
void writeMSB( char wert )
{
    writeLCR( readLCR() | 0x80 );   // DLAB-Bit im LCR setzen und Rest erhalten

    setAdr( 0,0,1 );     // Divisor-Latch MSB auswählen
    write( wert );       // Wert auf UART ausgeben

    writeLCR( readLCR() & 0x7F );   // DLAB-Bit wieder aus LCR Register entfernen
}


/*
 *   Interrupthandler für den UART
 *   ================
 */
set_int_serv() interrupt 4      // Interruptadresse/8 ergibt Interrupt Nummer
{
    // Überprüfen welcher Interrupt
    switch( readIIR() )
    {
        case 0: //MSR
        {
            //Modem Status Register wird nicht benötigt dies hier bleibt leer
        }
            break;

    //-----------------
        case 2: //THRE -> THR ist leer
        {
            //wenn THR leer dann THRleer = 1 setzen, Main programm kann neue Daten schicken
            THRleer = 1;
        }
            break;
        case 4: //Received Data Aviable (BRB read)
        {
            // wenn RBR eingelesen ist, dann leseflag auf 1 setzen, so das im Main Programm
            // das gelesene Byte ausgewertet werden kann
            greadData = readRBR();
            gmutex_read = 1;
        }



            break;
    //-------------------------------

        case 6: //Reciver Line Status (LSR read)
        {
            //einfachheitshalber wird LSR in der Globalen Variablen gLSR gespeichert
            //das könnte man eventuell später noch auswerten
            gLSR = readLSR();
        }
            break;
        case 255:   //Ende kein Interrupt
            break;
    }

    //---> muss der Interrupt eventuell auf dem Microcontroller gelöscht werden?!!!!!
}
```

8.1.4 UART-8250.h

```
/*
 *    Header für UART Baustein 8250
 *    -----------------------------
 *    2005.06
 *    Matthias Müller,  David Mayr,  Martin Junginger
 */


/*
 *   Diese Variablen hier werden wegen der Interrupt Service
 *   Routine (ISR) benötigt - (Globale Variablen)
 */

static char greadData;          // in der Variablen wird die Information
                                // gespeichert die vom Bus übernommen wird
static char gwriteData;         // die Information soll auf den Bus geschrieben werden
static char gLSR = 0;
static char THRleer = 1;        // solange gmutex_write = 0 kann in der Interrupt
                                // Routine THRE nicht ins THR geschrieben werden
static char gmutex_read = 0;    // solange gmutex-read = 0 wird nicht vom RBR gelesen

/*
 *   Bit-Namen
 *   =========
 */

sbit DISTR = 0x97;  // Data In  STRobe: Port 1 Bit 7
sbit DOSTR = 0x96;  // Data Out STRobe: Port 1 Bit 6

sbit ADR0  = 0xB3;  // Adressleitung 0: Port 3 Bit 3
sbit ADR1  = 0xB4;  // Adressleitung 1: Port 3 Bit 4
sbit ADR2  = 0xB5;  // Adressleitung 2: Port 3 Bit 5

sbit MR    = 0x92;  // Master Reset: Port 1 Bit 2


/*
 *   Funktionsprototypen
 *   ===================
 */

void initUART();                // UART initialisieren
void confUART();                // UART konfigurieren

void setAdr( char a2, char a1, char a0 ); // interne Funktion: Adresse an UART setzen
char read();                    // interne Funktion: Daten von  UART lesen
void write( char wert );        // interne Funktion: Daten auf  UART setzen
char readRBR();                 // TransHoldingRegister lesen
void writeTHR( char wert );     // THR schreiben
char readLCR();                 // LineControllRegister lesen
void writeLCR( char wert );     // LCR schreiben
char readLSR();             // LineStatusRegister (LSR) lesen
char readIIR();             // InterruptIdentifikationRegister (IIR) auslesen
void intEnable();           // Interrupt 0,1 in Interrupt enable register auswählen
void writeLSB( char wert ); // LSB der Baudraten-Einstellung
void writeMSB( char wert ); // MSB der Baudraten-Einstellung
```

8.1.5 TJA-1020.c

```
/*
 *    Treiber-Funktionen des TJA-1020 LIN-Transceiver
 *    -----------------------------------------------
 *    2005.06
 *    Matthias Müller,  David Mayr,  Martin Junginger
 */


#include "EXTRAS.h"      // Extra-Funktionen ( wait(), ... )
#include "UART-8250.h"   // Treiberfunktionen des UART 8250 einbinden
#include "TJA-1020.h"    // Treiberfunktionen des TJA-1020 LIN-Tranceiver


/*
 *   initTJA:  Init des TJA, sollte vor jedem Gebrauch des TJA aufgerufen werden
 *   =======
 */
void initTJA()
{
    NWAKE = 1;  // Pin NWAKE auf log. 1 setzen um NWAKE zu deaktivieren
}


/*
 *   startNSM: Start NormalSlopeMode. Bei Erfolg wird 1, sonst 0 zurückgegeben
 *   ========
 *   Einstellungen für den LSM:  pos. Flanke 5 ms an NLSP während TXD NICHT auf LOW ist (Normalzust./ kein Break)
 */
char startNSM()
{
    writeLCR( readLCR() & 0xBF );   // Bit 6 (BreakControl) im LCR ausschalten damit TXD=Normal

    NSLP = 1;   // pos. Flanke an NLSP-Pin erzeugen
    wait( 5 );  // ca. 5 ms warten, bis TJA reagiert
    NSLP = 0;   // NLSP-Pin zurücksetzen

    if( INH == 1 )  return 1;   // erfolgreich - ein SlopeMode liegt an
    else        return 0;   // nicht erfolgreich
}


/*
 *   startLSM: Start LowSlopeMode. Bei Erfolg wird 1, sonst 0 zurückgegeben
 *   ========
 *   Einstellungen für den LSM:  pos. Flanke an NLSP während TXD auf LOW ist
 */
char startLSM()
{
    writeLCR( readLCR() | 0x40 );   // Bit 6 (BreakControl) im LCR einschalten damit TXD=LOW


    NSLP = 1;   // pos. Flanke an NLSP-Pin erzeugen
    wait( 5 );  // ca. 5 ms warten, bis TJA reagiert
    NSLP = 0;   // NLSP-Pin zurücksetzen

    writeLCR( readLCR() & 0xBF );   // Bit 6 im LCR ausschalten damit TXD wieder verw. werden kann

    // Mit der nächsten pos. Flanke an TXD wird der TJA transmitter aktiviert.
    // Eigentlich sollte dies mit dem ersten zu sendenden Byte (Startbit) geschehen.
    // Darum wolen wir das Erzeugen der Flanke nicht an dieser Stelle vornehmen.

    if( INH == 1 )  return 1;   // erfolgreich - ein SlopeMode liegt an
    else        return 0;   // nicht erfolgreich
}


/*
 *   startSBM: Start StandByMode.
 *   ========
 */
char startSBM()
{
    // TODO: Implementierung
    return 0;
}


/*
 *   startSLM: Start SLeepMode.
 *   ========
 */
char startSLM()
{
    // TODO: Implementierung
    return 0;
}
```

8.1.6 TJA-1020.h

```
/*
 *    Header für LIN-Baustein TJA-1020
 *    --------------------------------
 *    2005.06
 *    Matthias Müller,  David Mayr,  Martin Junginger
 */

/*
 *   Bit-Namen
 *   =========
 */

sbit NSLP  = 0x95;  // NSLP  Port 1 Bit 5
sbit NWAKE = 0x94;  // NWAKE Port 1 Bit 4
sbit INH   = 0x93;  // INH   Port 1 Bit 3


/*
 *   Funktionsprototypen
 *   ===================
 */

void initTJA();      // Init des TJA

char startNSM();          // Start NormalSlopeMode (NSM)
char startLSM();          // Start LowSlopeMode (LSM)

char startSBM();          // Start StandByMode
char startSLM();          // Start SLeepMode
```

8.1.7 LIN-Frame.c

```
/*
 *    LIN-Frame
 *    ----------
 *    2005.06
 *    Matthias Müller,  David Mayr,  Martin Junginger
 */

#include "LIN-Frame.h"

/****** VORSICHT: Bytes werden mit dem LSB gestartet !!! ("verkehrtrum") ******/

/*
 *   calcParity: berechnet und setzt die beiden Parity Bits des Identifiers
 *   ==========
 */
void calcParity( struct LINframe frame )
{
    // Paritätsbit 0 bzw. ID7 ( ungerade Parität -> negiert )
    if ( !( (frame.ID & 0x40) ^      // ID-Bit 1
            (frame.ID & 0x10) ^      // ID-Bit 3
            (frame.ID & 0x08) ^      // ID-Bit 4
            (frame.ID & 0x04) ) )    // ID-Bit 5
        frame.ID |= 0x01;  // Bit 0/ID7 setzen
    else frame.ID &= 0xFE;  // Bit 0/ID7 löschen

    // Paritätsbit 1 bzw. ID6 ( gerade Parität -> nicht negiert )
    if  (   (frame.ID & 0x80) ^      // ID-Bit 0
            (frame.ID & 0x40) ^      // ID-Bit 1
            (frame.ID & 0x20) ^      // ID-Bit 2
            (frame.ID & 0x08) )      // ID-Bit 4
        frame.ID |= 0x02;  // Bit1/ID6 setzen
    else frame.ID &= 0xFD;  // Bit1/ID6 löschen
}



/*
 *   calcChksum():  Modulo256 Addition & Bitweise Invertierung des Ergebnisbytes
 *   =========
 */
void calcChksum( struct LINframe frame )
{
    frame.ID=1;
    // TODO
}



/*
 *   sendFrame: Versenden von LIN-Frames
 *   ==========
 */
void sendFrame( struct LINframe frame )
```

```
{
    char count;
    frame.ID=1;

    //berechnen des Prüfbytes


    //losschicken der Daten über den LIN-BUS, das ganze ist interruptgesteuert
    //hilfsvariable
    count = 0;

    //interruptgesteuertes versenden des Frames

    do
    {
        char THRleer; // TODO: blah
        if( ( THRleer == 1 ) && ( count == 0 ) )
        {
            //verschicken der praeabel
            // ???? writeTHR( 0x55 );
            THRleer = 0;
            count++;
        }

        //versenden des ID Bytes
        if(THRleer == 1 && count == 1)
        {
            // ??? writeTHR( frame.ID );
            THRleer = 0;
            count++;
        }

        //versenden der Datenbytes
        if(THRleer == 1 && count == 2)
        {
            //Die zu sendenden Daten ermitteln

        }

    } while(1);

}



/*
 *   receiveFrame: baut Frame zusammen
 *   ============
 */
char receiveFrame()
{
    // TODO: Implementation
    return( 0x00 );
}
```

8.1.8 LIN-Frame.h

```
/*
 *    Header f. LIN-Frame
 *    -------------------
 *    2005.06
 *    Matthias Müller,  David Mayr,  Martin Junginger
 */


/*
 *   Datentypen
 *   ==========
 */

static struct LINframe
{
    char ID;
    char daten[8];
    char chksum;
    char flags;      // Bit0 = Frame vollständig, Bit1 = chksum OK,
};
static struct LINfram frame;


/*
 *   Funktionsprototypen
 *   ===================
 */

void calcParity( struct LINframe frame );
void calcChksum( struct LINframe frame );
void sendFrame(  struct LINframe frame );
char receiveFrame();
```

8.1.9 LIN-Slave.c

```
/*
 *    LIN-Slave - LIN Slave-Task
 *    --------------------------
 *    2005.06
 *    Matthias Müller,  David Mayr,  Martin Junginger
 */


#include "REG515.H"      // Port- & Pin-Namen einbinden
#include "EXTRAS.h"      // Extra-Funktionen ( wait(), ... )
#include "C515-IO.h"     // IO-Funktionen für Peripherie des C515-MicroController (LEDs, Schalter, ...)
#include "UART-8250.h"   // Treiber-Funktionen & INT des UART 8250 Bausteins
#include "TJA-1020.h"    // Treiber-Funktionen des TJA-1020 LIN-Bausteins
#include "LIN-Frame.h"   // Struktur und Funktionen für LIN-Frames


/*
 *   init(): Initialisieren der Komponenten (UART & TJA)
 *   ====
 */
static void init()
{
    initUART();      // UART Baustein initialisieren
    confUART();      // UART mit Baudrate, Datenbit-Anzahl, Parität usw. konfigurieren

    initTJA();       // TJA LIN-Transceiver initialisieren
    startNSM();      // NormalSlopeMode des TJA starten
}


/*
 *   main(): Hauptfunktion des LIN-Slave
 *   ====
 */
static void main()
{
    init();          // UART Baustein initialisieren
}
```

8.1.10 Test-UART.c

```
/*
 *    Testprogramm für die serielle Komm. über den UART Baustein 8250
 *    ---------------------------------------------------------------
 *    2005.06
 *    Matthias Müller,  David Mayr,  Martin Junginger
 */


#include "REG515.H"
#include "EXTRAS.h"      // Extra-Funktionen ( wait(), ... )
#include "C515-IO.h"     // IO-Funktionen für Peripherie des C515-MicroController (LEDs, Schalter, ...)
#include "UART-8250.h"   // Treiberfunktionen des UART 8250 einbinden



/*
 *   testUART(): Hauptfunktion des UART-Testprogramms
 *   ========
 */
void TestUART()
{
    initUART();              // Baustein initialisieren
    confUART();              // Baustein mit Baudrate, Datenbit-Anzahl, Parität usw. konfigurieren

    while (1)
    {
        char i;
        for ( i=0; i<255; i++ )
        {
            writeTHR( i );   // Daten ins THR ablegen - wird automatich versendet
            wait( 3 );       // kurz warten
        }
    }

    setLEDs( readLCR() );    // LCR wieder auslesen und auf LEDs (Port4) ausgeben
}
```

8.1.11 Test-LIN.c

```
/*
 *    Testprogramm für den LIN-Bus / TJA-1220 LIN-Transceiver
 *    -------------------------------------------------------
 *    2005.06
 *    Matthias Müller,  David Mayr,  Martin Junginger
 */


#include "REG515.H"      // Port- & Pin-Namen einbinden
#include "EXTRAS.h"      // Extra-Funktionen ( wait(), ... )
#include "C515-IO.h"     // IO-Funktionen für Peripherie des C515-MicroController (LEDs, Schalter, ...)
#include "UART-8250.h"   // Treiber-Funktionen & INT des UART 8250 Bausteins
#include "TJA-1020.h"    // Treiber-Funktionen des TJA-1020 LIN-Bausteins
#include "LIN-Frame.h"   // Struktur und Funktionen für LIN-Frames


/*
 *   testLIN(): Hauptfunktion des LIN-Testprogramms
 *   =======
 */
void testLIN()
{
    initUART();      // UART Baustein initialisieren
    confUART();      // UART mit Baudrate, Datenbit-Anzahl, Parität usw. konfigurieren

    initTJA();       // TJA LIN-Transceiver initialisieren
    startNSM();      // NormalSlopeMode des TJA starten

    writeTHR( 0x55 );
}
```

8.2 Quellcode – PC-Alternative

8.2.1 UART.c

```
/*
 *    Treiberfunktionen des UART- / COM-Port
 *    ======================================================================
 *     NTA FH Isny          12.Info
 *     Modul:               Bussysteme & Interfaces
 *     Projektarbeit:       einfacher LIN-Slave (TJA1020 mit MAX232 an PC-COMport)
 *     Dozent/Betreuer:     Benno Gerum
 *    ----------------------------------------------------------------------
 *    David Mayr            <d.mayr(at)lunabox.de>
 *    Matthias Müller       <pullux(at)gmx.de>
 *    Martin Junginger      <ryoga-kun(at)gmx.de>
 *    ----------------------------------------------------------------------
 */



#include <stdio.h>
#include <unistd.h>
#include <asm/io.h>          // Port-Befehle u. Marcos einbinden

#include "UART.h"            // Funktionen und Konstanten des UART einbinden
#include "Debug.h"           // Debug-Funktionen einbinden



void UART_init()     // IO-Berechtigungen für UART freischalten
/*   =========       ---------------------------------------- */
{
    if ( ioperm(BASISADR, 8, 1)          // 8 Byte IO-Fenster ab Basisadr. freischalten
    {                                    // Bei Probleme Fehlerausgabe und Ende mit Status 1
        perror( "Fehler: ioperm() in initUART()" );
        exit( 1 );
    }
}



void UART_conf()     // Konfigurationsfunktion des UART (Baudrate, LCR, ...)
/*   =========       ------------------------------------------------------ */
{
    // Baudrate einstellen
    outb( ( inb(LCR) | 0x80 ) , LCR );  // DLAB-Bit (LCR-Bit7) setzen, um BRD-Register ansprechen zu können
    outb( 0x0C , BRDLSB );              // LSB des Baudratendivisors einstellen: für 9600Baud 0x0C; für 19200Baud 0x06
    outb( 0x00 , BRDMSB );              // MSB des Baudratendivisors einstellen: für 9600Baud 0x00; für 19200Baud 0x00
    outb( ( inb(LCR) & 0x7F ) , LCR );  // DLAB-Bit (LCR-Bit7) wieder zurücksetzen, um wieder auf THR, RDR und IER zugegreifen zu können

    // Schnittstellenparameter konfigurieren
    outb( 0x03 , LCR );                 // 8 Datenbits, 1 Stopbit, keine Parität

    // Interupt deaktivieren
    outb( (inb(MCR) & 0xF7) , MCR );    // Interupt deaktivieren -> kann unter Linux im Userspace nicht verwendet werden
    outb( 0x00 , IER );                 // Alle UART-Interrupt-Arten dektivieren (sollte eigentlich nicht extra nötig sein)

    // DTR-Leitung auf Ausgangszustand
    UART_setDTR( 0 );                   // DTR Bit auf 0
}



void UART_setDTR( _Bool wert ) // DTR-Leitung steuern -> MCR-Bit0
/*   ===========                ---------------------------------- */
{
    if ( wert ) {
        outb( ( inb(MCR) | 0x01 ) , MCR );  // DTR (=MCR-Bit0) auf 1 setzen
    } else {
        outb( ( inb(MCR) & 0xFE ) , MCR );  // DTR (=MCR-Bit0) auf 0 setzen
    }
}



void UART_setBreak( _Bool wert )     // Break auf TX-Leitung steuern -> LCR-Bit6
/*   =============                   ------------------------------------------ */
{
    if ( wert ) {
        outb( ( inb(LCR) | 0x40 ) , LCR );  // LCR-Bit6 (BreakControl) einschalten -> Break auf TX-Leitung
    } else {
        outb( ( inb(LCR) & 0xBF ) , LCR );  // LCR-Bit6 (BreakControl) ausschalten
    }
}
```

```
84  _Bool UART_checkBreak() .   // warten auf Break auf TX-Leitung
85  /*    ================ .   ---------------------------------- */
86  {
87      if ( inb(LSR) & 0x10 ) {.   // Wenn Break erkannt wurde (LSR-Bit4 = 1)
88  .       return( 1 );.   .   .   // TRUE (1) zurückgeben
89      } else {.   .   .   .   .   // sonst
90  .       return( 0 );.   .   .   // FALSE (0) zurückgeben
91  .   }
92  }
93
94
95
96  void UART_send( unsigned char wert ) .  // senden eines Zeichens sobald THR frei
97  /*   =========                      .  ---------------------------------------- */
98  {
99  .   while( !( inb(LSR) & 0x20 ) ).  // solange THR noch nicht frei/leer ist (LSR-Bit5 = 0)
100 .   {
101 .       usleep( 5 );.   .   // Warte (bei 19200 Bd dauert ein Bit ca. 52 µs)
102 .   }
103 .   outb( wert, THR );. .   // Zeichen in THR schreiben
104 }
105
106
107
108 unsigned char UART_receive() .  // Test-Schleife -> warte auf Empfang eines Zeiches und gib es zurück
109 /*           ============   .  ------------------------------------------------------------------------ */
110 {
111 .   while( !( inb(LSR) & 0x01 ) ).  // solange keine Daten im RDR (LSR-Bit0=0)
112 .   {
113 .       usleep( 5 );.   .   // Warte (bei 19200 Bd dauert ein Bit ca. 52 µs)
114 .   }
115 .   return( inb(RDR) );..   // empfangenes Zeichen zurückgeben
116 }
117
118
119
120 void UART_printRegs() . // Status- / Debug-Ausgaben
121 /*   ==============   . -------------------------- */
122 {
123 .   outb( ( inb(LCR) | 0x80 ) , LCR );. // DLAB-Bit (LCR-Bit7) setzen, um BRD-Register ansprechen zu können
124 .   printf( "\tBaudratendivisor(BRD): MSB=0x%02X  LSB=0x%02X\n", inb(BRDMSB), inb(BRDLSB) );.   // MSB und LSB des BaudratenDivisors ausgeben
125 .   outb( ( inb(LCR) & 0x7F ) , LCR );. // DLAB-Bit (LCR-Bit7) wieder zurücksetzen, um wieder auf THR, RDR und IER zugegreifen zu können
126 .   printf( "\tKontrollregister:      LCR=0x%02X  MCR=0x%02X  IER=0x%02X\n", inb(LCR), inb(MCR), inb(IER) );.   // LCR, MCR und IER ausgeben
127 .   printf( "\tStatusregister:        LSR=0x%02X  MSR=0x%02X  IIR=0x%02X\n", inb(LSR), inb(MSR), inb(IIR) );.   // LSR, MSR und IIR ausgeben
128 }
129
130
131
132 void UART_testDTR().// Test-Schleife für DTR-Leitung
133 /*   ============   .   ------------------------------ */
134 {
135 .   while( 1 )
136 .   {
137 .       printf( "\n\nDTR an: \n" );
138 .       UART_setDTR( 1 );
139 .       UART_printRegs();
140 .       sleep( 3 );
141
142 .       printf( "DTR aus:\n" );
143 .       UART_setDTR( 0 );
144 .       UART_printRegs();
145 .       sleep( 3 );
146 .   }
147 }
```

8.2.2 UART.h

```
/*
 *    Header für Konstanten und Treiberfunktionen des UART- / COM-Port
 *    ================================================================================
 *     NTA FH Isny          12.Info
 *     Modul:               Bussysteme & Interfaces
 *     Projektarbeit:       einfacher LIN-Slave (TJA1020 mit MAX232 an PC-COMport)
 *     Dozent/Betreuer:     Benno Gerum
 *    --------------------------------------------------------------------------------
 *    David Mayr            <d.mayr(at)lunabox.de>
 *    Matthias Müller       <pullux(at)gmx.de>
 *    Martin Junginger      <ryoga-kun(at)gmx.de>
 *    --------------------------------------------------------------------------------
 */



// Basis-IO-Adresse des UART (COM1=0x3F8, COM2=0x2F8)
#define BASISADR 0x3F8

// Register-Namen          //  | Adresse | R/W | Register | LCR.7 |
// --------------          //  | ------- | --- | -------- | ----- |
#define THR BASISADR+0     //  | Basis+0 | W   | THR (TX) |  0 !  |
#define RDR BASISADR+0     //  | Basis+0 | R   | RDR (RX) |  0 !  |
#define BRDLSB  BASISADR+0 //  | Basis+0 | R/W | BRD-LSB  |  1 !  |
#define BRDMSB  BASISADR+1 //  | Basis+1 | R/W | BRD-MSB  |  1 !  |
#define IER BASISADR+1     //  | Basis+1 | R/W | IER      |  0 !  |
#define IIR BASISADR+2     //  | Basis+2 | R   | IIR      |  -    |
#define LCR BASISADR+3     //  | Basis+3 | R/W | LCR      |  -    |
#define MCR BASISADR+4     //  | Basis+4 | R/W | MCR      |  -    |
#define LSR BASISADR+5     //  | Basis+5 | R   | LSR      |  -    |
#define MSR BASISADR+6     //  | Basis+6 | R   | MSR      |  -    |



/*
 *   Funktionsprototypen
 *   ===================
 */

void    UART_init();        // Init des UART (IO-Berechtigung)
void    UART_conf();        // Konfiguration des UART (Baudrate usw.)

void    UART_setDTR( _Bool wert );  // DTR-Leitung beeinflussen    (wert = [ 0 | 1 ])
void    UART_setBreak( _Bool wert );// Break auf TX-Leitung an/aus (wert = [ 0 | 1 ])
_Bool   UART_checkBreak();          // Prüfen, ob Break auf TX-Leitung erkannt wurde

void    UART_send( unsigned char wert );// senden eines Zeichens
unsigned char  UART_receive();      // warte auf Empfang eines Zeichens und gib es zurück

void    UART_printRegs();           // Ausgabe der Registerinhalte auf dem Bildschirm
void    UART_testDTR();             // Test-Schleife für DTR-Leitung

```

8.2.3 TJA1020.c

```
/*
 *    Treiber-Funktionen des TJA1020 LIN-Transceiver
 *    ================================================================================
 *     NTA FH Isny          12.Info
 *     Modul:               Bussysteme & Interfaces
 *     Projektarbeit:       einfacher LIN-Slave (TJA1020 mit MAX232 an PC-COMport)
 *     Dozent/Betreuer:     Benno Gerum
 *    --------------------------------------------------------------------------------
 *    David Mayr            <d.mayr(at)lunabox.de>
 *    Matthias Müller       <pullux(at)gmx.de>
 *    Martin Junginger      <ryoga-kun(at)gmx.de>
 *    --------------------------------------------------------------------------------
 */



#include "UART.h"          // Treiberfunktionen des UART einbinden
#include "TJA1020.h"       // Treiberfunktionen des TJA1020 LIN-Tranceivers einbinden


void TJA_send( unsigned char wert )    // senden eines Bytes
/*   ========                          ---------------------- */
{
    UART_send( wert );                 // verwende direkt UART-Funktion zum senden
}


unsigned char TJA_receive()    // warte auf Empfang eines Zeiches und gib es zurück
/*            ===========      ------------------------------------------------- */
{
    TJA_waitNoSyncBreak();
    return( UART_receive() );  // verwende hier direkt UART-Funktion zum empfangen
}


void TJA_setNSLP( _Bool wert ) // NSLP-Eingang des TJA1020 beeinflussen (HIGH/LOW)
/*   ===========               ------------------------------------------------- */
{
    UART_setDTR( wert );       // verwende UART-Funktion um NSLP durch DTR-Leitung zu beeinflussen
}


void TJA_disableTX( _Bool wert )   // Sync-Break Signal erzeugen (min. 13 Bitzeiten)
/*   =============                 ------------------------------------------------- */
{
    UART_setBreak( wert );         // verwende UART-Break-Bit um bei wert=1 TX auf LOW zu binden
}


void TJA_sendSyncBreak()   // Sync-Break Signal erzeugen (normal nur LIN-Master)
/*   =================     ------------------------------------------------- */
{
    TJA_disableTX( 1 );    // setze TX auf LOW
    usleep( 678 );         // min. 13 Bitzeiten => 13*(1/19200Bd) = ~678µs ;  13*(1/9600Bd) = 1355µs
    TJA_disableTX( 0 );    // setze TX zurück auf Normalbetrieb
}


_Bool TJA_checkSyncBreak()  // prüfen, ob Sync-Break Signal ankam
/*    ==================    ----------------------------------- */
{
    return( UART_checkBreak() );   // verwende UART-Funktion
}


void TJA_waitSyncBreak()   // Sync-Break Signal abwarten
/*   =================     ---------------------------- */
{
    //printDebug( "Warte auf Sync-Break " );
    while ( !TJA_checkSyncBreak() ) {  // Solange kein Break empfangen wurde
        usleep( 1 );                   // Warte (bei 19200 Bd dauert ein Bit ca. 52 µs)
        //printf( "." );
    }
```

```
}


void TJA_waitNoSyncBreak() // Sync-Break Signal abwarten
/*   ==================   ---------------------------- */
{
  //printDebug( "Warte bis Sync-Break vorbei" );
  while ( TJA_checkSyncBreak() ) {    // Solange kein Break empfangen wurde
    //usleep( 1 );                    // Warte (bei 19200 Bd dauert ein Bit ca. 52 µs)
    //printf( "." );
  }
}


void TJA_gotoNSMode() // Gehe in / starte NormalSlopeMode
/*   ==============   --------------------------------- */
{
  //  Einstellungen für den NSM:
  //  pos. Flanke 5 ms an NLSP während TXD NICHT auf LOW ist (Normalzust./ kein Break)

  TJA_disableTX( 0 );// setze TX zurück auf Normalbetrieb

  TJA_setNSLP( 1 );  // positive Flanke an NLSP-Pin erzeugen
  usleep( 6000 );    // mindestens 5 ms warten bis TJA reagiert
  TJA_setNSLP( 0 );  // NLSP-Pin zurücksetzen
}


void TJA_gotoLSMode() // Gehe in / starte LowSlopeMode
/*   ==============   ------------------------------- */
{
  //  Einstellungen für den LSM:
  //  pos. Flanke an NSLP während TXD auf LOW ist

  TJA_disableTX( 1 );// setze TX auf (Dauer-) LOW

  TJA_setNSLP( 1 );  // positive Flanke an NLSP-Pin erzeugen
  usleep( 6000 );    // mindestens 5 ms warten bis TJA reagiert
  TJA_setNSLP( 0 );  // NLSP-Pin zurücksetzen

  TJA_disableTX( 0 );// setze TX zurück auf Normalbetrieb

  // Mit der nächsten pos. Flanke an TXD wird der TJA transmitter aktiviert.
  // Eigentlich sollte dies mit dem ersten zu sendenden Byte (Startbit) geschehen.
  // Darum wollen wir das Erzeugen der Flanke nicht an dieser Stelle vornehmen.
}


void TJA_testModes()  // Test-Schleife für Mode-Umschaltung
/*   =============   ------------------------------------ */
{
  while( 1 )
  {
    printf( "\n\nNormal Slope Mode aktiviert.\n" );
    TJA_gotoNSMode();
    sleep( 3 );

    printf( "\n\nLow    Slope Mode aktiviert.\n" );
    TJA_gotoLSMode();
    sleep( 3 );
  }
}
```

8.2.4 TJA1020.h

```
/*
 *   Header für Treiber-Funktionen des TJA1020 LIN-Transceiver
 *   ================================================================================
 *    NTA FH Isny      12.Info
 *    Modul:           Bussysteme & Interfaces
 *    Projektarbeit:   einfacher LIN-Slave (TJA1020 mit MAX232 an PC-COMport)
 *    Dozent/Betreuer: Benno Gerum
 *   --------------------------------------------------------------------------------
 *   David Mayr       <d.mayr(at)lunabox.de>
 *   Matthias Müller  <pullux(at)gmx.de>
 *   Martin Junginger <ryoga-kun(at)gmx.de>
 *   --------------------------------------------------------------------------------
 */



/*
 *  Funktionsprototypen
 *  ===================
 */

void          TJA_send( unsigned char wert );  // senden eines Bytes
unsigned char TJA_receive();                   // warte auf Empfang eines Bytes

void          TJA_setNSLP( _Bool wert );       // NSLP-Eingang des TJA1020 beeinflussen

void          TJA_disableTX( _Bool wert );     // TX-Deaktivierung (LOW) an/aus (nötig für Sync-Break)

void          TJA_sendSyncBreak();             // Sync-Break Signal erzeugen (normalerweise nur LIN-Master)
_Bool         TJA_checkSyncBreak();            // prüfen, ob Sync-Break Signal ankam
void          TJA_waitSyncBreak();             // Sync-Break Signal abwarten
void          TJA_waitNoSyncBreak();           // Sync-Break Signal abwarten

void          TJA_gotoNSMode();                // Start NormalSlope Mode (NSM)
void          TJA_gotoLSMode();                // Start LowSlope Mode (LSM)

void          TJA_testModes();                 // Test-Schleife für Mode-Umschaltung
```

8.2.5 LINframe.c

```
/*
 *    LIN-Frame
 *    ========================================================================
 *     NTA FH Isny          12.Info
 *     Modul:               Bussysteme & Interfaces
 *     Projektarbeit:       einfacher LIN-Slave (TJA1020 mit MAX232 an PC-COMport)
 *     Dozent/Betreuer:     Benno Gerum
 *    ------------------------------------------------------------------------
 *    David Mayr           <d.mayr(at)lunabox.de>
 *    Matthias Müller      <pullux(at)gmx.de>
 *    Martin Junginger     <ryoga-kun(at)gmx.de>
 *    ------------------------------------------------------------------------
 */



// ***** VORSICHT: Bytes werden mit dem LSB gestartet !!! (verkehrtrum) *****

#include <math.h>      // Mathematische Funktionen einbinden

#include "LINframe.h"  // LINframe-Definition (struct) und -Funktionen einbinden


unsigned char LIN_getID( LINframe *frame ) // Gibt den direkten Wert der ID aus dem ID-Byte zurück
/*            =========                    ------------------------------------------------------- */
{
    return( frame->ID & 0x3F );    // gib nur die 6 niederwerigen Bits aus dem ID-Byte zurück
}


unsigned char LIN_getLength( LINframe *frame ) // Ermittelt Anzahl Datenbytes aus dem ID-Byte
/*            =============                    ---------------------------------------------- */
{
    switch( (frame->ID & 0x30) >>4 )    // je nach Bit4 und Bit5 aus ID-Byte
    {
        case( 0 ):
        case( 1 ):
            return( 2 ); break;     // zwei Datenbytes
        case( 2 ):
            return( 4 ); break;     // vier Datenbytes
        case( 3 ):
        default:
            return( 8 ); break;     // acht Datenbytes
    }
}


_Bool LIN_parityOK( LINframe *frame ) // Prüft beiden Parity Bits des Identifiers
/*    ============                    ------------------------------------------- */
{
    if (
        // Prüfe, ob Paritätsbit 0 OK ist ( gerade Parität -> nicht negiert )
        (
            (   (frame->ID & 0x01) ^    // ID-Bit0 xor
                (frame->ID & 0x02) ^    // ID-Bit1 xor
                (frame->ID & 0x04) ^    // ID-Bit2 xor
                (frame->ID & 0x10)      // ID-Bit4
            )
            &&
            (    frame->ID & 0x40    )  // -> ID-Bit6 (Paritätsbit-0)
            )
        )
        &&
        // Prüfe, ob Paritätsbit 1 OK ist ( ungerade Parität -> negiert )
        (
            !(  (frame->ID & 0x02) ^    // ID-Bit1 xor
                (frame->ID & 0x08) ^    // ID-Bit3 xor
                (frame->ID & 0x10) ^    // ID-Bit4 xor
                (frame->ID & 0x20)      // ID-Bit5
            )
            &&
            !(   frame->ID & 0x80    )  // -> ID-Bit7 (Paritätsbit-1)
        )
    ) {
        return( 1 );    // Parität OK
    } else {
```

```
81      return( 0 );    // Parität Fehlerhaft
82    }
83  }
84
85
86
87  unsigned char LIN_calcChksum( LINframe *frame )    // Checksumme berechnen und zurückgeben
88  /*            ==============                       ------------------------------------- */
89  {
90    int i, tmpChksum;
91    tmpChksum = 0;
92
93    // Alle Datenbytes addieren
94    for ( i=0; i<=LIN_getLength(frame); i++) {
95      tmpChksum += frame->daten[i];
96    }
97
98    // Modulo256 und "invertieren" der summierten Datenbytes
99    tmpChksum %= 0xFF;
100   tmpChksum = 0xFF - tmpChksum;
101   return( tmpChksum );
102 }
103
104
105
106 _Bool LIN_chksumOK( LINframe *frame )  // Modulo256 Addition & Bitweise Invertierung des Ergebnisbytes
107 /*   =============                     ------------------------------------------------------------- */
108 {
109   // Wenn Checksumme im Frame gleich der selbst berecheten Checksumme ist, dann ...
110   if ( frame->chksum == LIN_calcChksum(frame) ) {
111     return( 1 );    // ... gib TRUE  (1) zurück
112   } else {          // sonst ...
113     return( 0 );    // ... gib FALSE (0) zurück
114   }
115 }
116
117
118
119 void LIN_setChksum( LINframe *frame )  // Checksume berechnen lassen und dem Frame hinzufügen
120 /*   =============                     ------------------------------------------------------ */
121 {
122   frame->chksum = LIN_calcChksum(frame);
123 }
124
125
126
127 void LIN_printFrame( LINframe *frame ) // Gibt Inhalt des Frames auf dem Bildschirm aus
128 /*   =============                     ------------------------------------------- */
129 {
130   unsigned char i;
131   printf( "ID: ", LIN_getID(frame)  );
132   printf( "Daten (%i Bytes):  ", LIN_getLength(frame)  );
133   for ( i=0; i<=LIN_getLength(frame); i++ ) {
134     printf( "0x%20X  ", frame->daten[i] );
135   }
136 }
```

8.2.6 LINframe.h

```
/*
 *   Header für LIN-Frame
 *   ====================================================================
 *    NTA FH Isny          12.Info
 *    Modul:               Bussysteme & Interfaces
 *    Projektarbeit:       einfacher LIN-Slave (TJA1020 mit MAX232 an PC-COMport)
 *    Dozent/Betreuer:     Benno Gerum
 *   --------------------------------------------------------------------
 *   David Mayr            <d.mayr(at)lunabox.de>
 *   Matthias Müller       <pullux(at)gmx.de>
 *   Martin Junginger      <ryoga-kun(at)gmx.de>
 *   --------------------------------------------------------------------
 */



/*
 *  Frame-Struktur
 *  ==============
 */

typedef struct
{
    unsigned char ID;
    unsigned char daten[8];
    unsigned char chksum;
    unsigned char flags;    // Bit 0 = Frame vollständig, Bit 1 = chksum OK,
} LINframe;



/*
 *  Funktionsprototypen
 *  ===================
 */

unsigned char  LIN_getID( LINframe *frame );      // Gibt den direkten Wert der ID aus dem ID-Byte zurück
unsigned char  LIN_getLength( LINframe *frame );  // Ermittelt Anzahl Datenbytes aus dem ID-Byte

_Bool          LIN_parityOK( LINframe *frame );   // Prüft beiden Parity Bits des Identifiers

_Bool          LIN_chksumOK( LINframe *frame );   // Checksumme nachberechen und Prüfen
unsigned char  LIN_calcChksum( LINframe *frame ); // Checksumme berechnen und setzen
void           LIN_setChksum( LINframe *frame );  // Checksumme berechnen und setzen

void           LIN_printFrame( LINframe *frame ); // Gibt Inhalt des Frames auf dem Bildschirm aus

```

8.2.7 LINslave.c

```
/*
 *   LIN-Slave
 *   ========================================================================
 *    NTA FH Isny  .   12.Info
 *    Modul:  .  .   Bussysteme & Interfaces
 *    Projektarbeit:  einfacher LIN-Slave (TJA1020 mit MAX232 an PC-COMport)
 *    Dozent/Betreuer: Benno Gerum
 *   ------------------------------------------------------------------------
 *   David Mayr  .  .   <d.mayr(at)lunabox.de>
 *   Matthias Müller  <pullux(at)gmx.de>
 *   Martin Junginger  <ryoga-kun(at)gmx.de>
 *   ------------------------------------------------------------------------
 */



#include <stdio.h>

#include "Debug.h"  .   // Debug-Funktionen einbinden
#include "LINframe.h"   // LINframe-Definition (struct) und -Funktionen einbinden
#include "TJA1020.h"    // Treiberfunktionen des TJA1020 LIN-Tranceivers einbinden


void LIN_startSlave()
{
  .   int i;
  .   LINframe frame;


  .   while( 1 ) {

  .   .   // Warte auf Sync-Break
  .   .   TJA_waitSyncBreak();

  .   .   // Falls danach kein Sync-Field (0x55) empfangen wird -> Abbruch
  .   .   if ( TJA_receive() != 0x55 ) {
  .   .   .   // printDebug( "Break empfangen, aber kein folgendes Sync-field!\n" );
  .   .   .   continue;
  .   .   } else {
  .   .   .   printDebug( "OK - Break UND Sync-field empfangen.\n" );
  .   .   }

  .   .   // Falls noch ein Break ankam (Fehlerfall) -> Abbruch des Frame-Empfang
  .   .   //if ( TJA_checkSyncBreak() ) { continue; }

  .   .   // Empfange erstes Byte (ID-Byte)



  .   .   frame.ID = TJA_receive();

  .   .   // Prüfe Parität - Falls ID-Parität falsch -> Abbruch des Frame-Empfang
  .   .   if ( !LIN_parityOK( &frame ) ) {
  .   .   .   printf( "Paritaet falsch! - ID-Laenge=%i , ID=%X , ID-Byte=%X\n",
  .   .   .   .   LIN_getLength(&frame), LIN_getID(&frame), frame.ID );
  .   .   .   continue;
  .   .   }

  .   .   // Aktion abhängig vom ID:
  .   .   switch( LIN_getID(&frame) )
  .   .   {

  .   .   .   /* VORSICHT: Framelänge (in Bytes) hängt von Bit4 und Bit5 der ID ab!
  .   .   .    *
  .   .   .    *   0x00 - 0x1F  ->  2 Byte Daten
  .   .   .    *   0x20 - 0x2F  ->  4 Byte Daten
  .   .   .    *   0x30 - 0x3F  ->  8 Byte Daten ( 0x3C-0x3F reserviert )
  .   .   .    */

  .   .   .   // ID 0x3F bzw. 63 dez. ist reserviert für zukünftige Erweiterungen
  .   .   .   case(0x3F):
  .   .   .   .   printDebug( "ID 0x3F / 63 empfangen.\n" );
  .   .   .   break;

  .   .   .   // ID 0x3E bzw. 62 dez. ist reserviert für benutzerdefinierte Erweiterungen
  .   .   .   case(0x3E):
  .   .   .   .   printDebug( "ID 0x3E / 62 empfangen.\n" );
  .   .   .   break;

```

```
81          // ID 0x3D bzw. 61 dez. ist für Diagnose-Daten
82          case(0x3D):
83              printDebug( "ID 0x3D / 61 empfangen.\n" );
84          break;
85
86          // ID 0x3C bzw. 60 dez. ist für Diagnose-Daten
87          case(0x3C):
88              printDebug( "ID 0x3C / 60 empfangen.\n" );
89          break;
90
91          // alle übrigen IDs ( 0x00-0x3B bzw. 0-59 ) sind normale Frames
92
93
94
95          // ---> SENDEN von (8 Byte) Daten ...
96          case(0x30):
97              for ( i=0; i<=255; i+16 ) {
98                  TJA_send( i );
99              }
100             TJA_send( LIN_calcChksum(&frame) );
101         break;
102
103
104         // ---> EMPFANGEN von (8 Byte) Daten ...
105         case(0x33):
106             for ( i=0; i<LIN_getLength(&frame); i++ ) {
107                 frame.daten[i] = TJA_receive();
108             }
109             frame.chksum = TJA_receive();
110             if ( LIN_chksumOK(&frame) ) {
111                 LIN_printFrame(&frame);
112             } else {
113                 printDebug( "Frame mit fehlerhafter Checksumme empfangen und verworfen." );
114             }
115         break;
116
117
118
119         // ---> EMPFANGEN von (2Byte) Daten ...
120         case(0x01):
121             frame.daten[0] = TJA_receive();
122             frame.daten[1] = TJA_receive();
123             frame.chksum = TJA_receive();
124             if ( LIN_chksumOK(&frame) ) {
125                 LIN_printFrame(&frame);
126             } else {
127                 printDebug( "Frame mit fehlerhafter Checksumme empfangen und verworfen." );
128             }
129         break;
130
131         // ---> SENDEN von (2Byte) Daten ...
132         case(0x00):
133             TJA_send( 0xFF );
134             TJA_send( 0x00 );
135             TJA_send( LIN_calcChksum(&frame) );
136         break;
137
138         // wenn eine Nachricht gar nicht beachtet wird nur Debug-Meldung ausgeben
139         default:
140             printDebug( "Frame mit unbekannter ID empfangen und verworfen." );
141         break;
142     }
143 }
144 }
```

8.2.8 LINslave.h

```
1  /*
2   *   Header für LIN-Slave
3   *   ====================================================================
4   *    NTA FH Isny      12.Info
5   *    Modul:           Bussysteme & Interfaces
6   *    Projektarbeit:   einfacher LIN-Slave (TJA1020 mit MAX232 an PC-COMport)
7   *    Dozent/Betreuer: Benno Gerum
8   *   --------------------------------------------------------------------
9   *   David Mayr       <d.mayr(at)lunabox.de>
10  *   Matthias Müller  <pullux(at)gmx.de>
11  *   Martin Junginger <ryoga-kun(at)gmx.de>
12  *   --------------------------------------------------------------------
13  */
14
15
16
17  /*
18  *   Funktionsprototypen
19  *   ===================
20  */
21
22  void    LIN_startSlave();    // Starte LIN-Slave in Endlosschleife
```

8.2.9 _SLAVEtest.c

```
/*
 *    TEST-Datei: LIN
 *    ================================================================================
 *     NTA FH Isny  .    12.Info
 *     Modul: . .   .    Bussysteme & Interfaces
 *     Projektarbeit: .  einfacher LIN-Slave (TJA1020 mit MAX232 an PC-COMport)
 *     Dozent/Betreuer:. Benno Gerum
 *    --------------------------------------------------------------------------------
 *    David Mayr .  .    <d.mayr(at)lunabox.de>
 *    Matthias Müller .  <pullux(at)gmx.de>
 *    Martin Junginger . <ryoga-kun(at)gmx.de>
 *    --------------------------------------------------------------------------------
 */


#include "UART.h"    .    // Treiberfunktionen des UART einbinden
#include "TJA1020.h"     // Treiberfunktionen des TJA1020 LIN-Tranceivers einbinden
#include "LINslave.h"    // LINslave einbinden


int main()
{
    UART_init();     .    // UART initialisieren
    UART_conf();     .    // und konfigurieren

    TJA_gotoNSMode();    // LIN-Transceiver aktivieren (NormalSlope Mode)

    LIN_startSlave();    // Starte LIN Slave (Endlosschleife)

    return( 0 );
}
```

8.2.10 Debug.c

```
/*
 *    Debug Hilfs-Funktionen
 *    ======================================================================
 *     NTA FH Isny  .    12.Info
 *     Modul: . .   .    Bussysteme & Interfaces
 *     Projektarbeit: . einfacher LIN-Slave (TJA1020 mit MAX232 an PC-COMport)
 *     Dozent/Betreuer:. Benno Gerum
 *    ----------------------------------------------------------------------
 *    David Mayr .  .    <d.mayr(at)lunabox.de>
 *    Matthias Müller . <pullux(at)gmx.de>
 *    Martin Junginger .<ryoga-kun(at)gmx.de>
 *    ----------------------------------------------------------------------
 */



void printDebug( char text[] ) .// Gibt Debug-Meldung aus, sofern aktiviert
/*   ======================== .---------------------------------------- */
{
.   char doDebug;
.   doDebug = 1;.   // Debug-Meldungen aktivieren (1) / deaktivieren (0)

.   if ( doDebug == 1 ) {
.   .   printf( "\n *** DEBUG: %s", text );
.   }
}


void printDebug2( char text[] ) .   // Gibt Debug-Meldung aus, sofern aktiviert
/*   ======================== .---------------------------------------- */
{
.   char doDebug;
.   doDebug = 1;.   // Debug-Meldungen aktivieren (1) / deaktivieren (0)

.   if ( doDebug == 1 ) {
.   .   printf( "%s", text );
.   }
}
```

8.2.11 Debug.h

```
/*
 *    Header für Debug Hilfs-Funktionen
 *    ======================================================================
 *     NTA FH Isny  .    12.Info
 *     Modul: . .   .    Bussysteme & Interfaces
 *     Projektarbeit: . einfacher LIN-Slave (TJA1020 mit MAX232 an PC-COMport)
 *     Dozent/Betreuer:. Benno Gerum
 *    ----------------------------------------------------------------------
 *    David Mayr .  .    <d.mayr(at)lunabox.de>
 *    Matthias Müller . <pullux(at)gmx.de>
 *    Martin Junginger .<ryoga-kun(at)gmx.de>
 *    ----------------------------------------------------------------------
 */



/*
 *   Funktionsprototypen
 *   ===================
 */

void .  printDebug( char text[] );. .   // Debug-Meldung ausgeben, sofern aktiviert
void .  printDebug2( char text[] );..   // Debug-Meldung ausgeben, sofern aktiviert
```