07.06.2023
Version 1.05

Anwenderschnittstelle USV-Slave

Projektbezeichnung	Unmanned Surface Vehicle	
Projektleiter	Jörg Grabow	
Verantwortlich	Stefan Franke	
Erstellt am	13.11.2020	
Zuletzt geändert	07.06.2023	
Bearbeitungsstand	i.B.	in Bearbeitung vorgelegt fertig gestellt
Dokumentenablage	https://github.com/Joe-Grabow/USV/tree/main/00 doc/00 Bussystem	

Änderungsverzeichnis

Änderung			geänderte Kapitel	Beschreibung	Autor	neuer Zustand
Nr.	Datum	Version				
1	13.11.2020	1.00	-	Startversion	Fr.	f.g.
2	25.11.2020	1.01	Übertragungs-protokoll	Änderung der Framelänge auf n+7	Fr.	f.g.
3	03.12.2020	1.02	Steckverbinder J5	Funktionstabelle J5	Gr.	f.g.
4	04.12.2020	1.03	Handshake	Beschreibung Handshake	Gr.	f.g.
5	09.12.2020	1.04	Error Status Byte	Byte für Fehlerzustände	Gr.	i.B.
6	17.02.2021	1.05	Protokoll	Baud = 250.000, Framelänge Tab. 5	Gr.	f.g.

in Bearbeitung (i.B.)
Vorlage (Vg.)
fertig gestellt (f.g.)

07.06.2023
Version 1.05

Einleitung

Um Daten auf dem RS485-Bus zu schreiben oder zu lesen, ist am USV-Slave eine physische Datenschnittstelle und ein Übertragungsprotokoll notwendig. Da eine UART-Schnittstelle auf den meisten Mikrocontrollern zu finden ist, wurde diese Schnittstelle für die Datenübertragung ausgewählt. Zur sicheren Datenübertragung wird auf der UART-Schnittstelle ein framebasierendes Übertragungsprotokoll realisiert. Dieses Dokument beschreibt die UART-Schnittstelle und das zugehörige Übertragungsprotokoll, um eine störungsfreie Kommunikation zwischen dem USV-Slave und der User-Unit zu gewährleisten.

Die UART-Schnittstelle

Die UART-Schnittstelle (Universal Asynchronous Receiver Transmitter) beschreibt eine Schnittstelle zwischen zwei Kommunikationsgeräten. Die UART-Schnittstelle ist als eine Untermenge des RS-232 Standard zu verstehen. Standardmäßig werden für die Schnittstelle zwei Datensignale benötigt. Es können zur Datenflusskontrolle jedoch noch zwei weitere Steuerleitungen genutzt werden. Die Schnittstelle ist in der maximalen Hardwarebeschaltung für den Vollduplexbetrieb ausgelegt. Das bedeutet, beide Geräte können unabhängig voneinander gleichzeitig Daten senden und empfangen. Der logische Pegel ist dabei nicht definiert und von der konkreten Hardware abhängig (verwendetes Treiber-IC). Auch Steckverbinder und -belegung sind nicht definiert. Die Pinbeschreibung ist in folgender Tabelle dargestellt.

Pin	Pin-Name	Daten- / Steuerleitung	Beschreibung	I/O
TX	Transmit Data	D	Pin für gesendete Daten	O
CTS	Clear to Send	S	Pin für das Empfangen der Sendeerlaubnis	I
RX	Receive Data	D	Pin für empfangene Daten	I
RTR/RTS	Redy to Receive / Request to Send	S	Pin zum Senden der Empfangserlaubnis	O

Tab. 1: UART-Pinbeschreibung

Damit zwei UART-Treiber miteinander kommunizieren können, müssen immer beide Daten- und Steuerleitungen gekreuzt werden. Das heißt, TX ist jeweils RX verschaltet und RTS jeweils mit CTS.
Die einzige feste Definition der UART-Schnittstelle, bezieht sich auf den zeitlichen Verlauf der Datenübertragung. Dabei sind die Baudrate, die Anzahl der Datenbits, der Stoppbits und das Paritätsbit variabel. Diese Werte sind bei beiden Kommunikationsgeräten gleich einzustellen, da sonst keine Daten sicher übertragen werden können. In der nachfolgenden **Abb. 1** ist der zeitliche Verlauf eines UART-Frames mit 8 Datenbits, einem Paritätsbits und einem Stoppbit dargestellt.

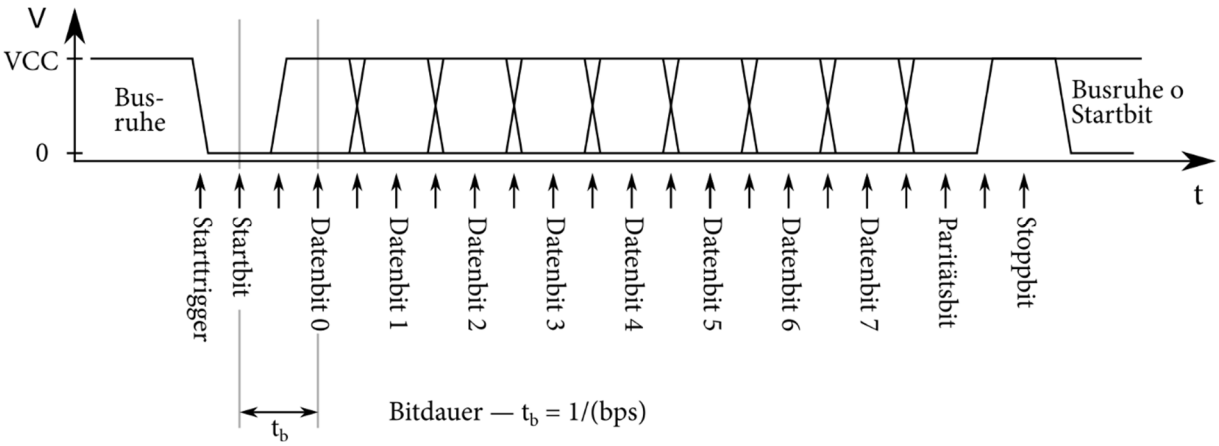

Abb. 1: zeitlicher Ablauf eines UART-Frame

Der Frame besteht aus einem Startbit gefolgt von den Datenbits, einem optionalen Paritätsbit und dem Stoppbit. Nach einer Übertragung eines Frames erfolgt eine Busruhe. Es kann aber auch nach dem Stoppbit sofort wieder mit einem Startbit für das nächste zu übertragende Frame gesendet werden. Die Bruttodatenübertragung ist bei der UART-Schnittstelle um 50% größer als die Nettodatenübertragung, und deswegen anderen seriellen Schnittstellen unterlegen. Ein Vorteil ist jedoch die einfache Implementierung sowie die Datenübertragung mit nur zwei Datenleitungen.

UART-Konfiguration

Die UART-Konfiguration wird vom USV-Slave festgelegt und ist auf dem User-Unit Bord exakt gleich einzustellen.

Typ	Wert	Beschreibung
Baudrate	250000	Bits pro Sekunde
Bits	8	Anzahl der Bits pro UART-Frame
Parität	ungerade	Parität (gerade, ungerade, 1 oder 0)
Stoppbits	1	Länge der Stoppbits

Tab. 2: UART-Konfiguration

Übertragungsprotokoll

Frameaufbau

Das Übertragungsprotokoll beschreibt den Übertragungsablauf zum USV-Slave, um Daten auf den RS485-Bus zu schreiben oder Daten vom RS485-Bus zu lesen. Die Daten, die auf den RS485-Bus geschrieben werden dürfen sind abhängig von Typ der an dem USV-Slave angeschlossenen User-Unit. Die Steuerung erfolgt über eine *slavespezifische ID*, die im Übertragungsprotokoll mitgesendet wird. In einem Frame sind nach dem Startbyte die slavespezifische ID, die Adresse für die Daten, die Framelänge, die eigentlichen Daten, die CRC-8 Summe gebildet aus dem Polynom 0xD5 und das Endbyte untergebracht. Zusätzlich zu einem Frame gibt es noch eine Kurzquittung. Diese versendet nur der USV-Slave als Bestätigung („acknowledge“ **ACK** oder „not acknowledge“ **NACK**). Der Frameaufbau ist in folgender Tabelle dargestellt:

Wert		Beschreibung		Länge in Bytes
0xA5		Startbyte		1
0xXX		slavespezifische ID		1
0x8XXX	0x4XXX	Schreiben (4bits) + Adresse (12bits)	Lesen (4bits) + Adresse (12bits)	2
0xXX		Framelänge = n+7		1
0xXX		n Datenbytes		n
0xXX		CRC-8 von den Datenbytes; Polynom 0xD5		1
0xA6		Endbyte		1

Tab. 3: Frameaufbau

Daten schreiben

Um Daten an den USV-Slave zu schreiben, werden die Daten in dem oben beschrieben Frame eingebettet und anschließend über die UART-Schnittstelle versendet. Sind alle Daten richtig empfangen und dürfen die Daten an die jeweilige Adresse geschrieben werden, antwortet der USV-Slave über die UART-Schnittstelle anschließend mit einem ACK. Bei Fehlern antwortet der USV-Slave mit einem NACK. In der folgenden **Tab. 4**: wird der Schreibablauf für das Übertragungsprotokoll dargestellt:

TX	RX	Beschreibung	Länge in Bytes
0xA5	-	Startbyte	1
0xXX	-	slavespezifische ID	1
0x8XXX	-	Daten Schreiben (4bits) + Adresse (12bits)	2
0xXX	-	Framelänge = n+7	1
0xXX	-	Daten	n

0xXX	-	CRC-8 von den Datenbytes	1
0xA6	-	Endbyte	1
-	0xA1 ∨ 0xA2	Antwort vom USV-Slave • 0xA1 – ACK • 0xA2 – NACK	1

Tab. 4: Daten schreiben

Daten lesen

Um Daten vom USV-Slave zu lesen, wird wie beim Senden von Daten ein Frame an den USV-Slave gesendet. Statt den Daten werden stattdessen die Anzahl der zu lesenden Bytes gesendet. Im Anschluss antwortet der USV-Slave mit den Daten, die wiederum in einem Frame eingebettet sind. In der nachfolgenden **Tab. 5**: ist der Leseablauf für das Übertragungsprotokoll dargestellt.

Byte	TX	RX	Beschreibung	Länge in Bytes
1	**0xA5**	-	Startbyte	1
2	**0xXX**	-	slavespezifische ID	1
3 … 4	**0x4XXX**	-	Daten Lesen (4bits) + Adresse (12bits)	2
5	**0x08**	-	Framelänge = 8	1
6	**0xXX**	-	Datenlängenanfrage = n	1
7	**0xXX**	-	CRC-8 von der Datenlängenanfrage	1
8	**0xA6**	-	Endbyte	1
1	-	0xA5	Startbyte bei erfolgreicher Anfrage	1
2	-	0xXX	slavespezifische ID	1
3 … 4	-	0x0XXX	Null Bits (4bits) + Leseadresse (12bits)	2
5	-	0xXX	Framelänge = n+7	1
6 … n+6	-	0xXX	Daten (LSB … MSB)	n
n+7	-	0xXX	CRC-8 von den Datenbytes	1
n+8	-	0xA6	Endbyte	1

Tab. 5: Daten lesen

Sind keine Daten an der angeforderten Adresse oder ist die slavespezifische ID falsch, antwortet der USV-Slave mit einem NACK und geht anschließend wieder in die Busruhe.

Bsp.: Anfrage nach dem Kurswinkel (Adresse 0x00C, Länge zwei Bytes)

1	2	3	4	5	6	7	8
Startbyte	**Slave ID**	**Addr (LSB)**	**Addr (MSB)**	**Framelänge**	**Datenlänge**	**CRC-8**	**Endbyte**
0xA5	0xXX	0x0C	0x40	0x08	0x02	0x7F	0xA6

Hardwareschnittstelle

Das Slave-Modul stellt über J5 eine Hardwareschnittstelle für eigene Anwendungen bereit. Auf diesem Steckverbinder liegen neben den oben besprochenen Datenleitungen für die serielle Kommunikation noch weitere Signale sowie zwei Spannungen zur Versorgung der eigenen spezifischen Anwendung. Tab. 6 gibt einen Überblick über die Signale und die Steckerbelegung.

Pin	Pin-Name	Beschreibung	I/O
1	+5V	Power +5 V	O
2	GND	Masse	O
3	+3.3V	Power 3.3 V	O
4	GND	Masse	O
5	VCC	teilt dem Slave mit, welcher Logikpegel (3.3 V oder 5.0 V) verwendet wird	I
6	RX	Daten vom Slave an den Bus	I
7	TX	Daten vom Bus an den Slave	O
8	RTS	Slave möchte Senden (Request to send = Aufforderung zum Senden)	O
9	CTS	warte auf Sendeerlaubnis (Clear to send = Erlaubnis zum Senden erteilt)	I
10	IRQ	Interrupt Meldung vom Slave	I
11	ERROR 1	Fehlermeldung über Buszustand	O
12	ERROR 2	Fehlermeldung über Buszustand	O

Tab. 6: Pinbelegung des Slave-Steckerbinders J5

Handshake DTE-DCE

Die RS-232 Schnittstelle wurde ursprünglich dazu geschaffen, um Computerterminals (DTE - data terminal equipment) an langsame Modems (DCE - data communication equipment) anzuschließen.
Der USV-Slave besitzt einen eigenständigen Prozessor, welcher mit dem RS485-Bus kommuniziert und auf denen der Anwender nur über die RS232-Schnittstelle (siehe Hardwareschnittstelle J5) zugreifen kann. Dieser Prozessor hat die Funktion des früheren Terminals. Die anwenderspezifische Hardware übernimmt die Funktion des früheren Modems.
Bei den damaligen DFÜ-Geschwindigkeiten konnte das Modem die Daten nicht so schnell versenden, wie das Terminal sie liefern konnte. Deshalb gibt es im RS232-Standard zwei Steuerleitungen, mit denen der Datenfluss gesteuert (also bei Bedarf gebremst) werden kann.
Die dafür verwendeten Leitungen heißen RTS und CTS. Am Terminal gibt es einen RTS-Ausgang und einen CTS-Eingang (siehe J5). Am Modem gibt es einen RTS-Eingang und einen CTS-Ausgang (Abb. 2).
Wenn Hardwarehandshake verwendet wird, dann aktiviert das Terminal zuerst die RTS-Leitung (request to send - darf ich senden?) und fragt damit beim Modem an, ob es bereit ist Daten zu empfangen. Wenn das Modem bereit ist, aktiviert es nun seinerseits die CTS-Leitung (clear to send - bin bereit). Erst nun sendet der Sender auf der TXD-Datenleitung asynchron die oben beschriebene Datenbytes mit Start- und Stopbits. Ist das Modem irgendwann nicht mehr in der Lage die vom Terminal eintreffenden Daten schnell genug weiterzuverarbeiten, deaktiviert es die Leitung CTS. Daraufhin unterbricht das Terminal den Datentransport solange, bis CTS wieder vom Modem aktiviert wird. Auf diese Art und Weise verschafft das Modem sich Pausen für die interne Verarbeitung der Daten.

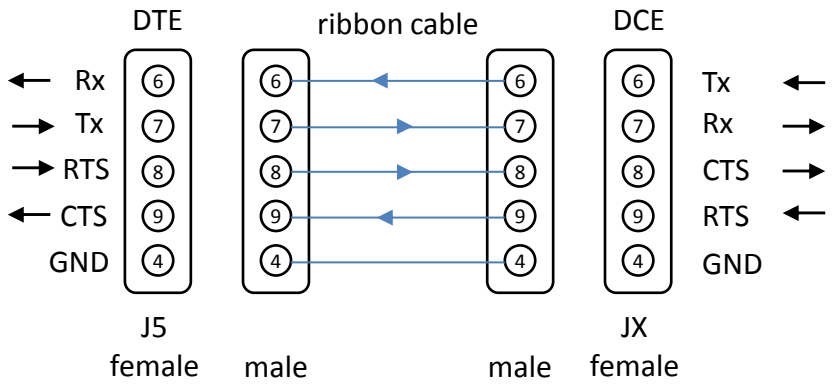

Abb. 2: Darstellung der seriellen Verbindung zwischen beiden Prozessoren

Fehlerzustände

Um als externer Beobachter Fehlerzustände innerhalb des Bussystems und der Hardware verfolgen zu können, werden unterschiedliche Schutzmechanismen implementiert. Jedes aktive Slave-Modul sendet neben seinen eigentlichen Daten, am Schluss des Datenblocks ein ***Error Status Byte (ESB)***. In diesem werden unterschiedliche Systemzustände kodiert (siehe Abb. 3. und Tab. 7). Der Busmaster überwacht alle ESB's der jeweiligen Slave-Module und bildet daraus ein ***globales Error Status Byte (GESB).***

Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
ESB Bit	UB3	UB2	UB1	UB0	3.3V	5V	BORF	WDRF
Initial Value	0	0	0	0	0	0	0	0

Abb. 3: Aufbau eines Error Status Byte (ESB)

Bit	Bit Name	Beschreibung	R/W
0	WDRF	Dieses Bit wird gesetzt „1“, wenn ein Watchdog Reset auftrat.	W
1	BORF	Dieses Bit wird gesetzt „1“, wenn ein Brown-out Reset auftrat.	W
2	5V	Dieses Bit wird gesetzt „1“, wenn +5V um 20% abweicht.	W
3	3.3V	Dieses Bit wird gesetzt „1“, wenn +3.3V um 20% abweicht.	W
4	UB0	Hier können eigene Fehlerzustände kodiert werden.	W
5	UB1	z.B. GPS-Koordinaten nicht plausibel, Winkeldaten außerhalb Wertebereich usw.	W
6	UB2	dto.	W
7	UB3	dto.	W

Tab. 7: Kodierung der Fehlerzustände im ESB