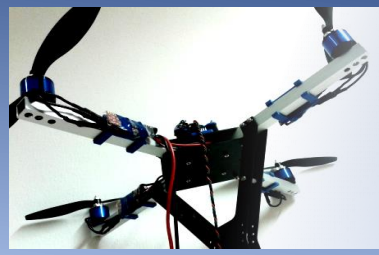
QCS-Einführungskurs

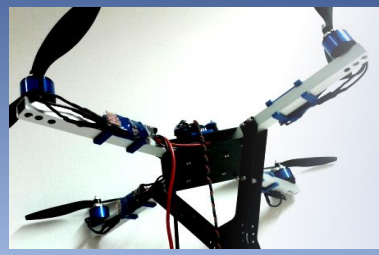
Serielle Kommunikation



Inhalt

Umfang: ca. 2 Zeitstunden

- Was ist serielle Datenübertragung?
- USART Theorie
- Serielle Übertragung mit dem QCS
- HTerm – Einführung
- Aufgaben

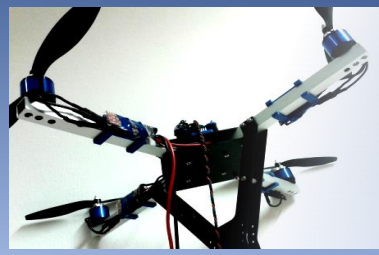


Serielle Datenübertragung

Was ist serielle Datenübertragung?

- Bits einer Nachricht werden **nacheinander auf einer Leitung** übertragen
- Bedeutung und Anordnung der Bits über **UART / USART** definiert
- Verschiedene Standards für verschiedene Einsatzgebiete
- Typische Vertreter im Alltag: RS-232, Ethernet, USB
- Kommunikation auf Platinenebene über **I²C** **auch beim QCS!**

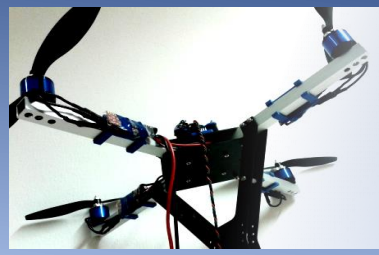
D-Sub Stecker einer seriellen Schnittstelle (RS232, 9-polig)



USART (Theorie)

Begriffliches:

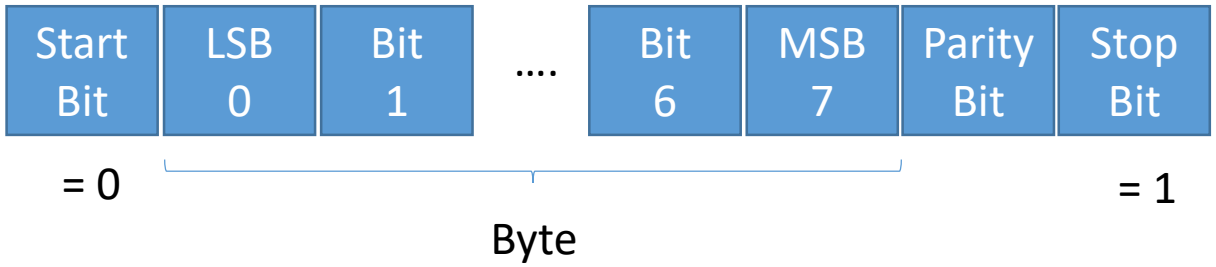
- **UART:** Universal Asyncronous Receiver Transmitter
- **USART:** Universal Syncronous Asyncronous Receiver Transmitter
- UART ist Teil von USART
- Bezeichnet elektronisches Bauteil und Schnittstelle

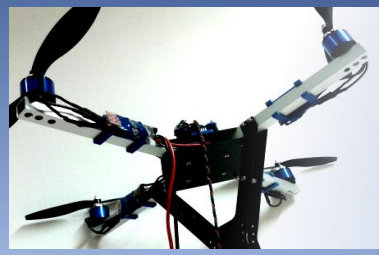


USART (Theorie)

Theorie:

- **Synchron:** Mit Clock oder Handshake – Verfahren
- **Asynchron:** Keine Clock -> billig, einfach und verbreitet
- Unterstützt **Voll-Duplex:** Senden und Empfangen gleichzeitig
- **Leitungen:** Tx (zum Senden), Rx (zum Empfangen), CLK (Synchronisation), RTS (Request to Send, Sendeanfrage) , CTS (Clear to Send, Sendeerlaubnis)
- **Parity Bit** zur einfachen Fehlererkennung





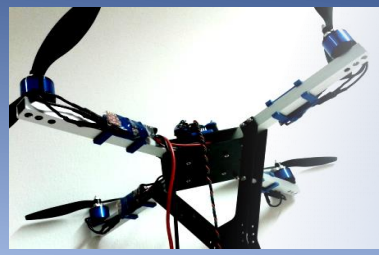
USART (Theorie)

Besonderheiten **Asynchron**:

- Startbit und Stopbit legen Frame und Synchronisationsstart und -ende fest
- maximale Framelänge ist begrenzt, um Synchronisationsverlust zu vermeiden
- **Wichtig:** Baudrate muss für Synchronisation bekannt sein: Entspricht hier der Bitrate

Anmerkung:

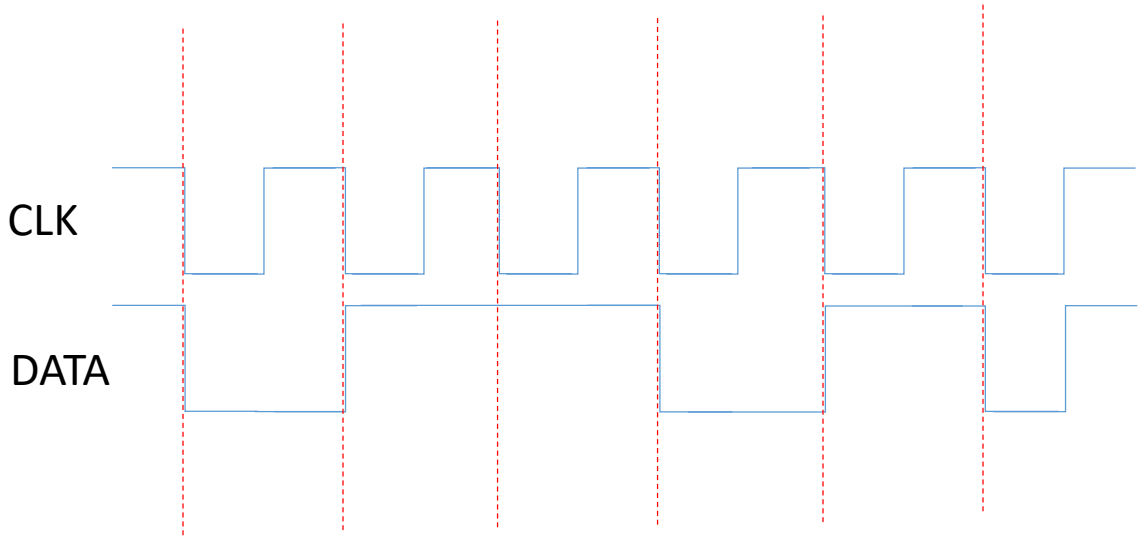
- Baudrate = Symbole / s oder Zeichen / s — Schrittgeschwindigkeit
- Bitrate = Bit / s — Datenübertragungsrate

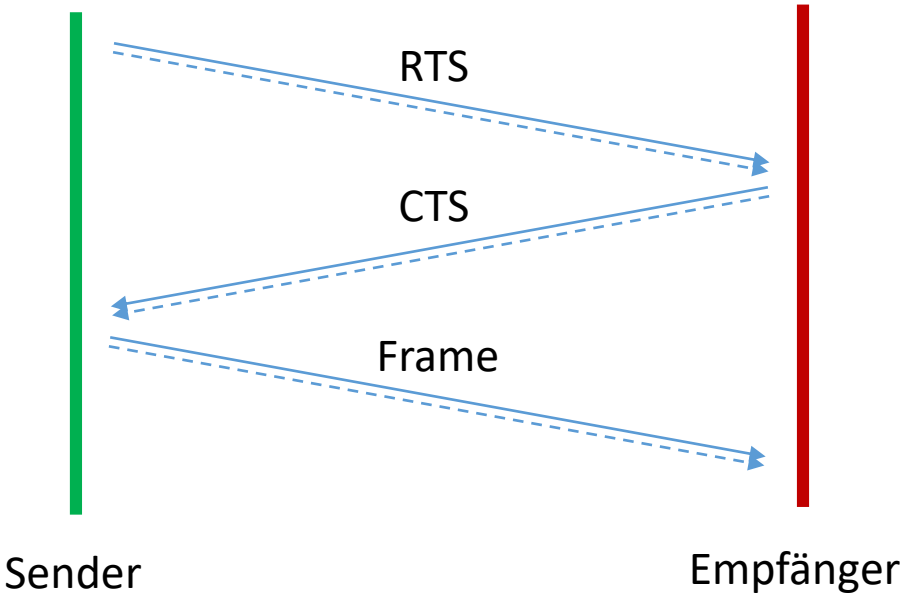


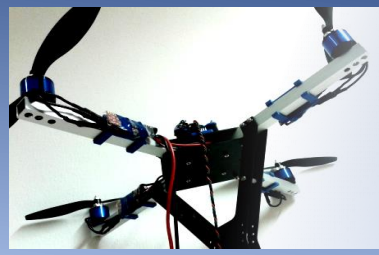
USART (Theorie)

Besonderheiten Synchron:

- **Takt** (Clock, CLK) gibt Übertragungsgeschwindigkeit vor
- (Hardware-) **Handshake** (RTS, CTS) für Koordination von Sender und Empfänger





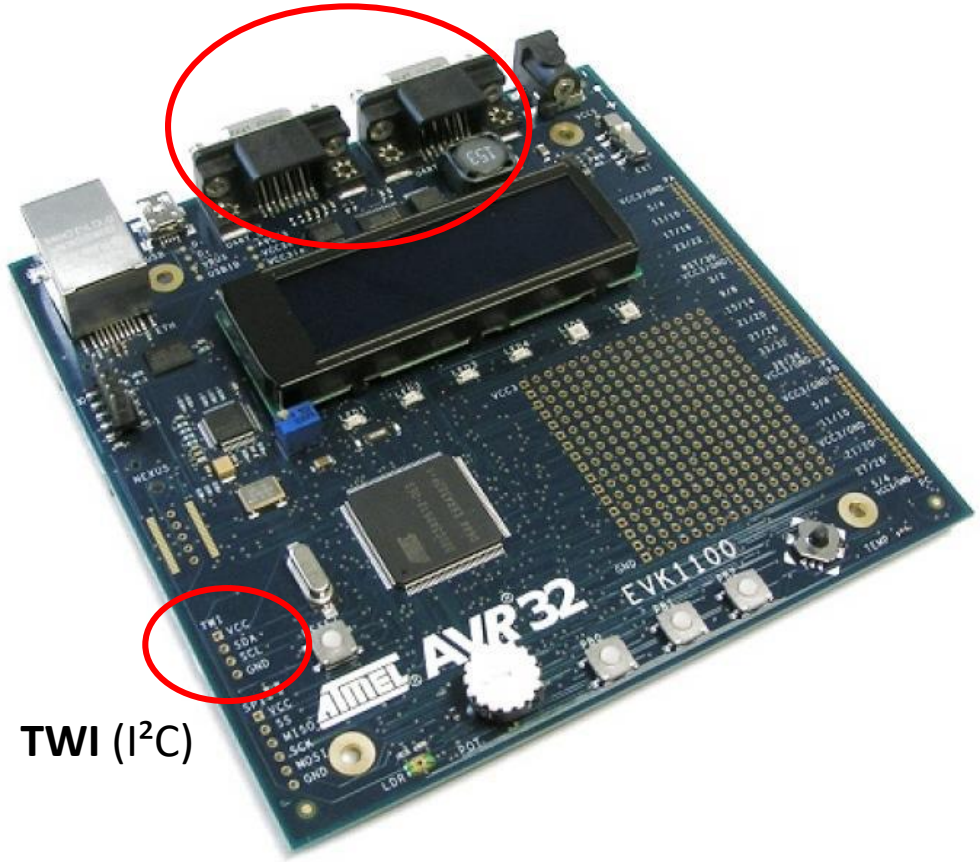


Serielle Kommunikation beim QCS

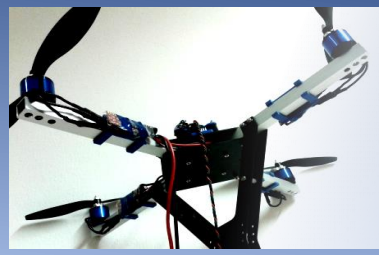
Einsatz serieller Kommunikation beim QCS:

- Telemetrie und Telekommandos
- Ansteuern und Auslesen von Sensoren
- Debugging

USART RS-232
Schnittstellen



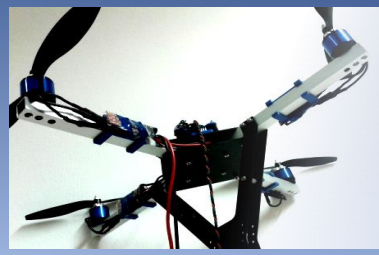
TWI (I^2C)



Serielle Kommunikation beim QCS

Senden und Empfangen von Nachrichten über RS-232

- Verfügbare Funktionen aus der Bibliothek **libemq.a** :
  - `void` **`USART_write`**`(char* line)`
    Schickt einen String via serieller USART - Schnittstelle
  - `dataStream` **`usart_read`**`()`
    Ließt einen Datenstrom von der seriellen Schnittstelle (USART 0) aus. Die Daten liegen dabei im Puffer des DMA und können von da direkt verarbeitet werden. Nachdem die Daten verarbeitet wurden, müssen diese mit der Funktion `void` **`usart_release`**`()` freigegeben werden, damit der Speicherbereich vom DMA für neue Daten genutzt werden kann. Andernfalls kommt es zu Problemen! Dieses Verfahren ist effizienter, als ein vorheriges Kopieren.
  - `void` **`usart_release`**`()`
    Gibt den zuvor gelesenen Speicherbereich des DMAs wieder frei. Zunächst ist mit `dataStream` **`usart_read`**`()` ein Datenstrom zu lesen und nach Wunsch zu verarbeiten. Anschließend muss der Speicherbereich mit `void` **`usart_release`**`()` freigegeben werden. Wird der Speicher nicht freigegeben, läuft der Puffer des DMAs voll und es kommt zu Problemen, die es zu vermeiden gilt.



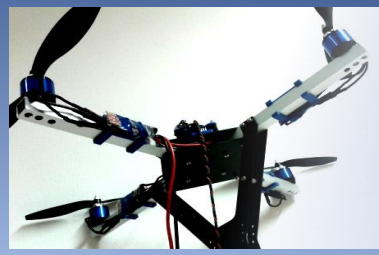
Serielle Kommunikation beim QCS

Exkurs: Direct Memory Access (DMA)

- Auf deutsch: „Direkter Speicher Zugriff“ -> Ohne Kontrolle durch die CPU
- Direkter Austausch von Daten zwischen Speicher und Peripherie im Hintergrund
- Ohne nennenswerte Belastung der CPU

Vorgehensweise:

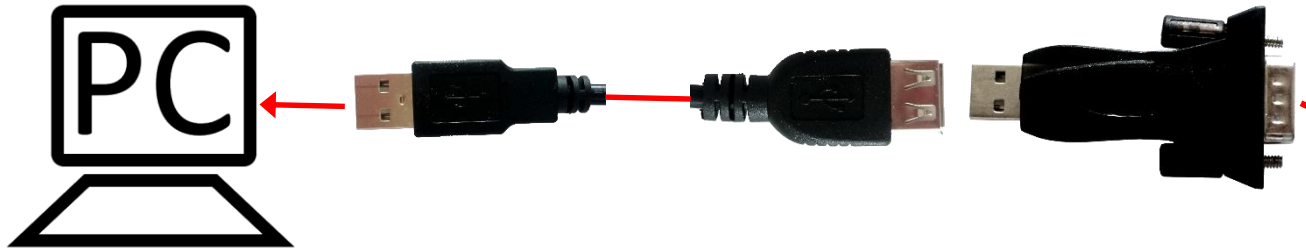
1. Datenlänge und Start-Speicherort übergeben
2. Übertrage ein Datenwort
3. Inkrementiere Speicherort
4. Dekrementiere Restdatenlänge
5. Warte bis Interface wieder bereit
6. Wiederhole 2 – 5 bis Restdatenlänge = 0

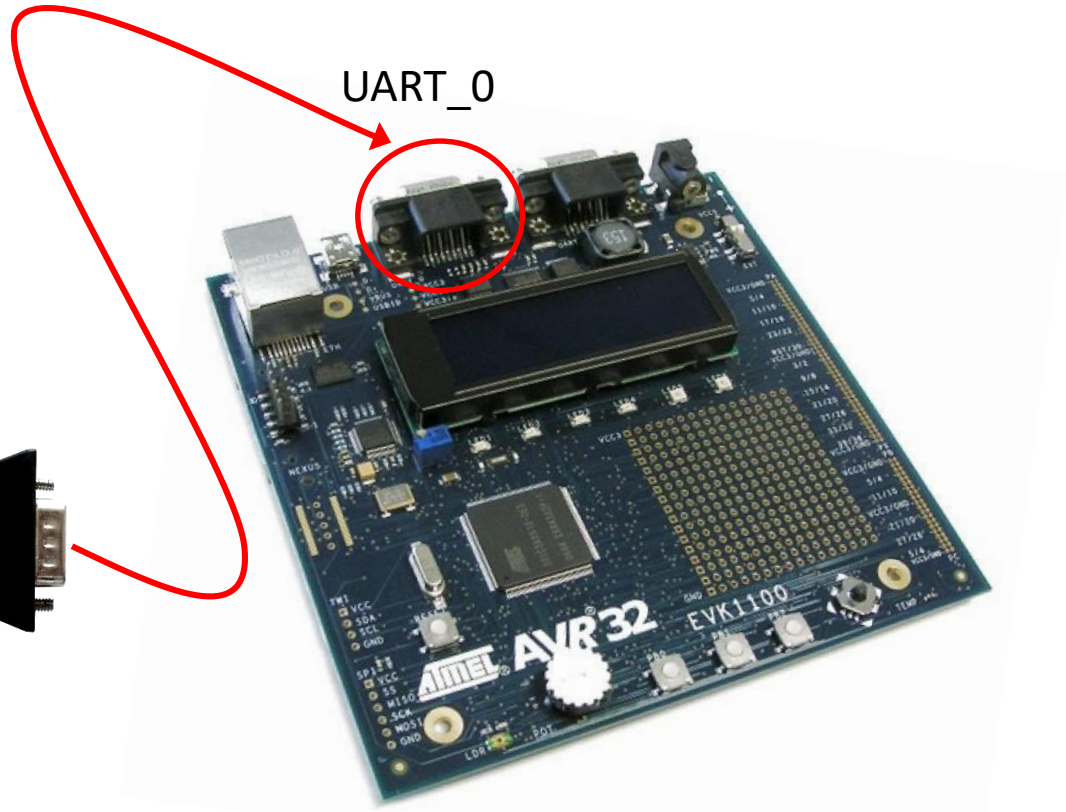


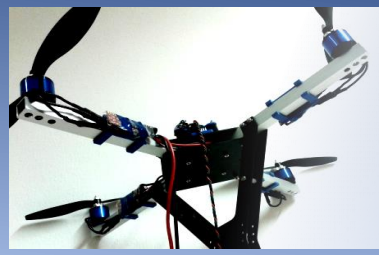
Serielle Kommunikation beim QCS

- Kommunikation mit PC über COM – Ports
- HTerm: Programm zum Darstellen und Senden von Nachrichten

<u>Anschluss des EVK1100 an den PC:</u>





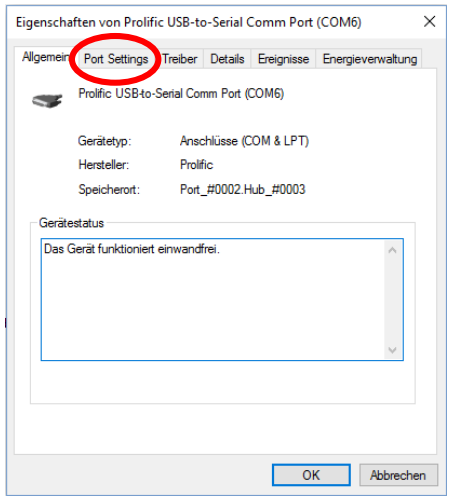


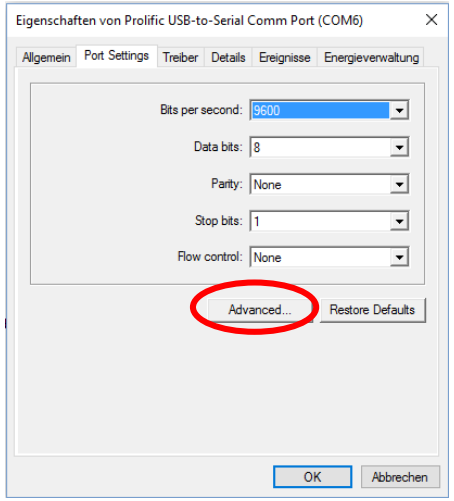
Serielle Kommunikation beim QCS

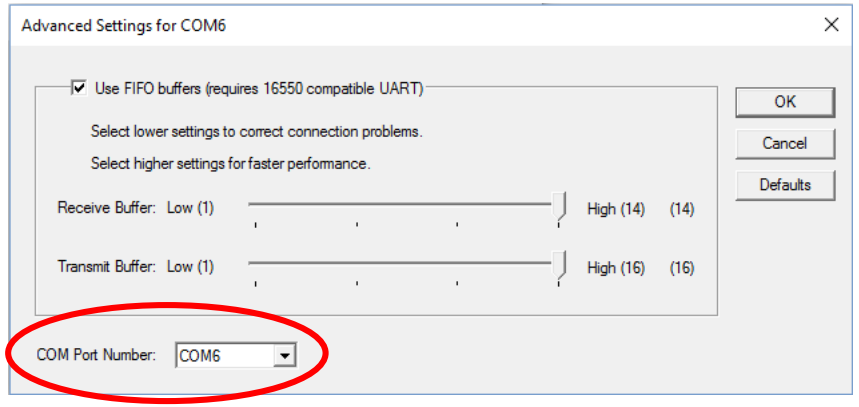
Bestimmen des COM Ports des RS-232 Converters

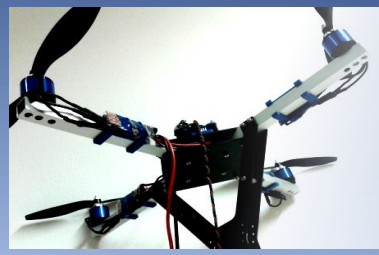
Vorgehensweise:

1. Gerätemanager öffnen
2. RS-232 Converter finden (Unter Anschlüsse COM & LPT)
3. Mit Rechtsklick Eigenschaften des Gerätes öffnen
4. Unter Port Settings -> Advanced… -> COM Port Number





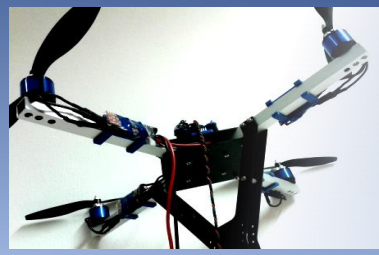




HTerm

- Kostenloses Terminalprogramm für die serielle Schnittstelle RS-232
- Viele Optionen zum Senden und Auslesen von Nachrichten:
  - COM-Port
  - Baudrate
  - Darstellung als Dezimal, Hexadezimal, Binär und ASCII
  - Newline-Codes

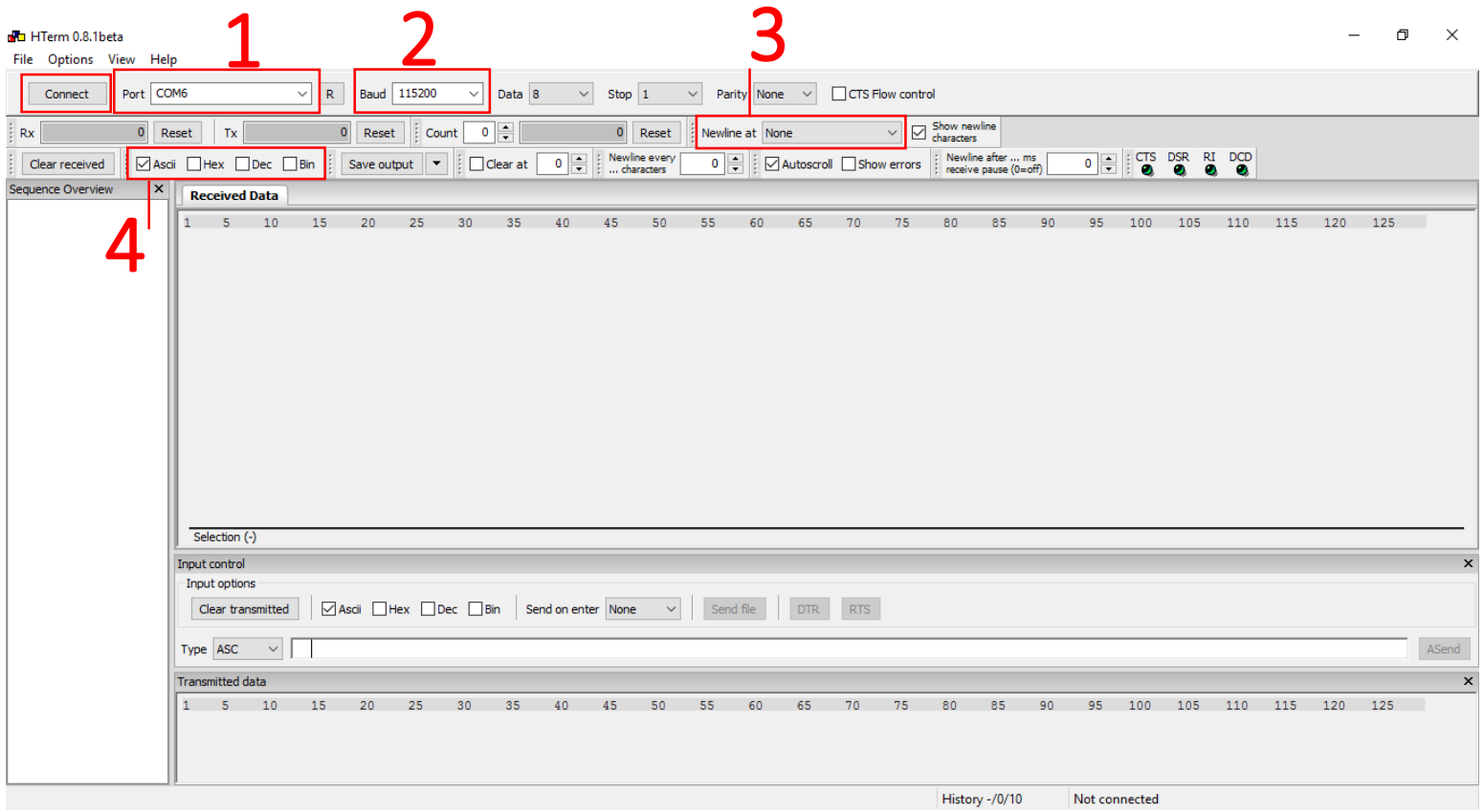
HTerm

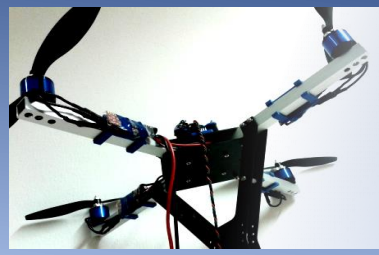


HTerm

Konfiguration für Kommunikation mit EVK1100:

1. COM – Port von RS-232 Converter einstellen
2. Baudrate: **57600**
3. Newline at: **LF**
4. Format: **ASCII**
5. Verbinden über **Connect**
6. Empfange Nachrichten von EVK11000





Auslesen von Nachrichten

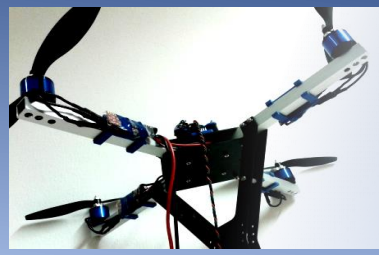
Mit `void` **`usart_read`** Nachrichten empfangen und verarbeiten

- `dataStream` **`usart_read()`** liefert Nachrichten vom USART in ein `dataStream` struct.

- `dataStream` besteht aus:
  `char*` `data` = Nachrichteninhalt
  `unsigned int` `size` = Anzahl der Zeichen in der Nachricht

- `data[i]` liefert den i-ten Eintrag der Nachricht.

  Beispiel:
  `hallo: [0]=h, [1]=a, [2]=l, [3]=l, [4]=o`



Wiederholung zu sprintf

sprintf:

- Konkatenation von Strings und Variablen
- Beispiel:

```
char line[20];
int wert = 3;
sprintf(line, „Wert ist :%i ", wert);
USART_write(line);
```

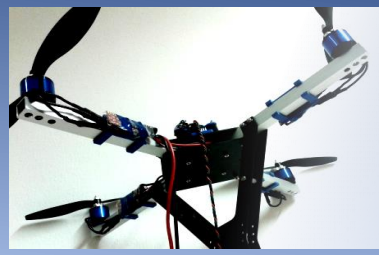
- Sprintf Parameter siehe Tabelle

%	Bedeutung	Beispiel
d or i	Signed decimal integer	392
u	Unsigned decimal integer	7235
o	Unsigned octal	610
x	Unsigned hexadecimal integer	7fa
X	Unsigned hexadecimal integer (uppercase)	7FA
f	Decimal floating point, lowercase	392.65
F	Decimal floating point, uppercase	392.65
e	Scientific notation (mantissa/exponent), lowercase	3.9265e+2
E	Scientific notation (mantissa/exponent), uppercase	3.9265E+2
g	Use the shortest representation: %e or %f	392.65
G	Use the shortest representation: %E or %F	392.65

Gute (sichere) Alternative:

```
snprintf(line, 20, „Wert ist :%i ", wert);
```

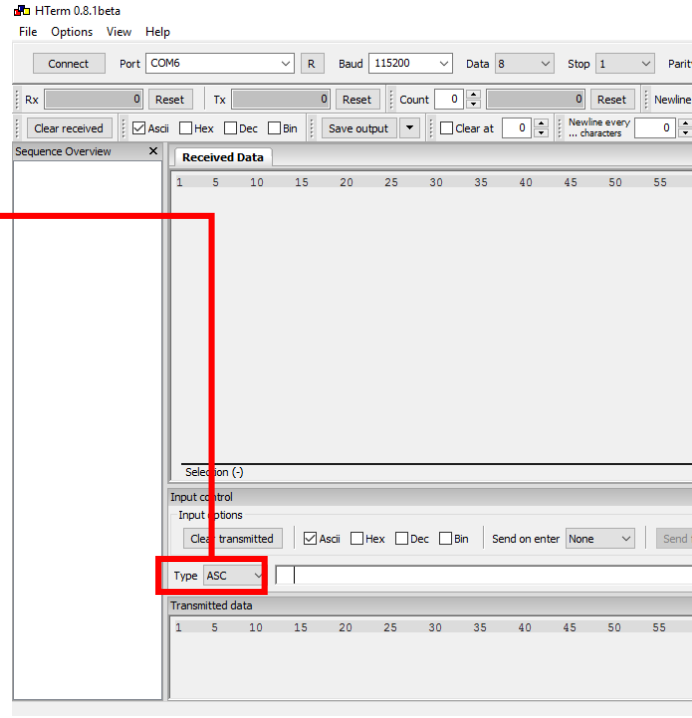
Maximal 20 Zeichen lang.



Senden von Nachrichten via HTerm:

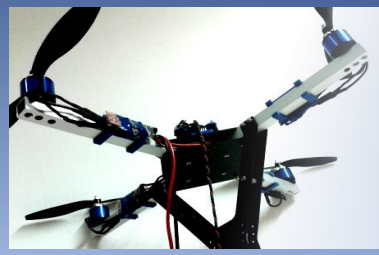
- Konvertierung des Eingabe – Textes ins Wunsch - Datenformat vor dem Senden

- **Optionen für Datenformat:**
  - Ascii
  - Hex
  - Dec
  - Bin



- **Achtung beim Versenden von Ascii – Zeichen:**

Eingabe wird vor dem Senden entsprechend der ASCII – Tabelle in Bytes umgewandelt, die auf der Empfängerseite in das gewünschte Format umgewandelt werden müssen.



Vom Ascii - Zeichen zur Ganzzahl

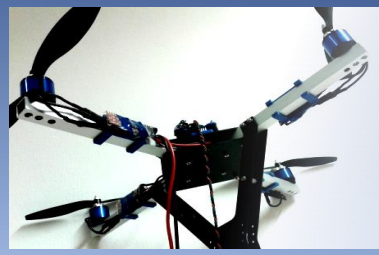
- **Problem:**

```
char c  = '0';
int i = c;
```

Was steht jetzt in `i` ?

ASCII - Tabelle

000	NUL	033	!	066	B	099	c	132	ä	165	Ñ	198	ã	231	þ
001	Start Of Header	034	"	067	C	100	d	133	à	166	ª	199	Ã	232	Þ
002	Start Of Text	035	#	068	D	101	e	134	å	167	º	200	╚	233	Ú
003	End Of Text	036	$	069	E	102	f	135	ç	168	¿	201	╔	234	Û
004	End Of Transmission	037	%	070	F	103	g	136	ê	169	®	202	╩	235	Ù
005	Enquiry	038	&	071	G	104	h	137	ë	170	¬	203	╦	236	ý
006	Acknowledge	039		072	H	105	i	138	è	171	½	204	╠	237	Ý
007	Bell	040	(	073	I	106	j	139	ï	172	¼	205	═	238	¯
008	Backspace	041	)	074	J	107	k	140	î	173	¡	206	╬	239	´
009	Horizontal Tab	042	*	075	K	108	l	141	ì	174	«	207	¤	240	-
010	Line Feed	043	+	076	L	109	m	142	Ä	175	»	208	ð	241	±
011	Vertical Tab	044	,	077	M	110	n	143	Å	176	░	209	Ð	242	‗
012	Form Feed	045	-	078	N	111	o	144	É	177	▒	210	Ê	243	¾
013	Carriage Return	046	.	079	O	112	p	145	æ	178	▓	211	Ë	244	¶
014	Shift Out	047	/	080	P	113	q	146	Æ	179	│	212	È	245	§
015	Shift In	048	0	081	Q	114	r	147	ô	180	┤	213	ı	246	÷
016	Delete	049	1	082	R	115	s	148	ö	181	Á	214	Í	247	¸
017	-- frei --	050	2	083	S	116	t	149	ò	182	Â	215	Î	248	°
018	-- frei --	051	3	084	T	117	u	150	û	183	À	216	Ï	249	¨
019	-- frei --	052	4	085	U	118	v	151	ù	184	©	217	┘	250	·
020	-- frei --	053	5	086	V	119	w	152	ÿ	185	╣	218	┌	251	¹
021	Negative Acknowledge	054	6	087	W	120	x	153	Ö	186	║	219	█	252	³
022	Synchronous Idle	055	7	088	X	121	y	154	Ü	187	╗	220	▄	253	²
023	End Of Transmission Block	056	8	089	Y	122	z	155	ø	188	╝	221	¦	254	■
024	Cancel	057	9	090	Z	123	{	156	£	189	¢	222	Ì	255	
025	End Of Medium	058	:	091	[	124	|	157	Ø	190	¥	223	▀		
026	Substitude	059	;	092	\	125	}	158	×	191	┐	224	Ó		
027	Escape	060	<	093	]	126	~	159	ƒ	192	└	225	ß		
028	File Seperator	061	=	094	^	127	⌂	160	á	193	┴	226	Ô		
029	Group Seperator	062	>	095	_	128	Ç	161	í	194	┬	227	Ò		
030	Record Seperator	063	?	096	`	129	ü	162	ó	195	├	228	õ		
031	Unit Seperator	064	@	097	a	130	é	163	ú	196	─	229	Õ		
032		065	A	098	b	131	â	164	ñ	197	┼	230	µ		



Vom Ascii - Zeichen zur Ganzzahl

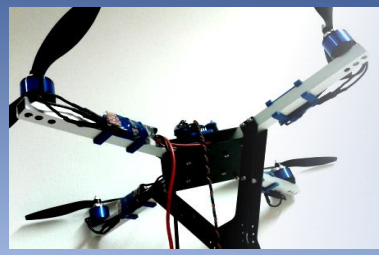
Konvertierung von char zu int:

- **Antwort aus der ASCII-Tabelle:**
  Das Zeichen ‘0‘ hat den „Index“ 48.

Daher steht in `i` die Zahl: 48

Somit ist ‘(‘ mal ‘(‘ = 1600
und ‘1‘ mal ‘1‘ = 2401

000	NUL	033	!	066	B	099	c	132	ä	165	Ñ	198	ã	231	þ
001	Start Of Header	034	"	067	C	100	d	133	à	166	ª	199	Ã	232	Þ
002	Start Of Text	035	#	068	D	101	e	134	å	167	º	200	╚	233	Ú
003	End Of Text	036	$	069	E	102	f	135	ç	168	¿	201	╔	234	Û
004	End Of Transmission	037	%	070	F	103	g	136	ê	169	®	202	╩	235	Ù
005	Enquiry	038	&	071	G	104	h	137	ë	170	¬	203	╦	236	ý
006	Acknowledge	039		072	H	105	i	138	è	171	½	204	╠	237	Ý
007	Bell	040	(	073	I	106	j	139	ï	172	¼	205	═	238	¯
008	Backspace	041	)	074	J	107	k	140	î	173	¡	206	╬	239	´
009	Horizontal Tab	042	*	075	K	108	l	141	ì	174	«	207	¤	240	-
010	Line Feed	043	+	076	L	109	m	142	Ä	175	»	208	ð	241	±
011	Vertical Tab	044	,	077	M	110	n	143	Å	176	░	209	Ð	242	‗
012	Form Feed	045	-	078	N	111	o	144	É	177	▒	210	Ê	243	¾
013	Carriage Return	046	.	079	O	112	p	145	æ	178	▓	211	Ë	244	¶
014	Shift Out	047	/	080	P	113	q	146	Æ	179	│	212	È	245	§
015	Shift In	048	0	081	Q	114	r	147	ô	180	┤	213	ı	246	÷
016	Delete	049	1	082	R	115	s	148	ö	181	Á	214	Í	247	¸
017	-- frei --	050	2	083	S	116	t	149	ò	182	Â	215	Î	248	°
018	-- frei --	051	3	084	T	117	u	150	û	183	À	216	Ï	249	¨
019	-- frei --	052	4	085	U	118	v	151	ù	184	©	217	┘	250	·
020	-- frei --	053	5	086	V	119	w	152	ÿ	185	╣	218	┌	251	¹
021	Negative Acknowledge	054	6	087	W	120	x	153	Ö	186	║	219	█	252	³
022	Synchronous Idle	055	7	088	X	121	y	154	Ü	187	╗	220	▄	253	²
023	End Of Transmission Block	056	8	089	Y	122	z	155	ø	188	╝	221	¦	254	■
024	Cancel	057	9	090	Z	123	{	156	£	189	¢	222	Ì	255	
025	End Of Medium	058	:	091	[	124	|	157	Ø	190	¥	223	▀		
026	Substitude	059	;	092	\	125	}	158	×	191	┐	224	Ó		
027	Escape	060	<	093	]	126	~	159	ƒ	192	└	225	ß		
028	File Seperator	061	=	094	^	127	⌂	160	á	193	┴	226	Ô		
029	Group Seperator	062	>	095	_	128	Ç	161	í	194	┬	227	Ò		
030	Record Seperator	063	?	096	`	129	ü	162	ó	195	├	228	õ		
031	Unit Seperator	064	@	097	a	130	é	163	ú	196	─	229	Õ		
032		065	A	098	b	131	â	164	ñ	197	┼	230	µ		



Vom Ascii - Zeichen zur Ganzzahl

Konvertierung von char zu int:

- **Die Lösung:**

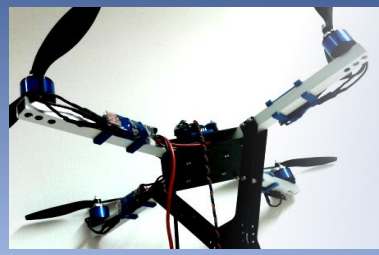
Der Index des Zeichens muss zum Zahlenwert umgerechnet werden!

Die Verschiebung der Zahlen in der ASCII- Tabelle beträgt 48.
Daher lautet die Formel zur Umrechnung von char zu int:

```
char c = …;
int i = c – 48;
```

Oder einfach:

```
int i = c – '0';
```



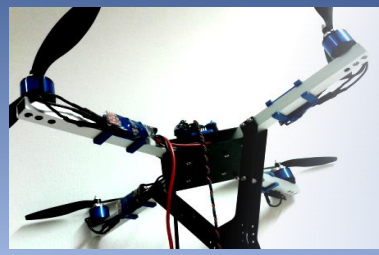
Aufgaben

Notwendige Hardware:

- EVK1100
- QCS / QCSF
- Mikro USB Kabel für Strom und zum Flashen
- USART – USB Adapter

Notwendige Software:

- AVR Studio 32 installiert (mit Tool Chain und FLIP Treiber)
- Projekt-Code (Framework + Lösung) und Library: EMQ_Framework, EMQ_QCS, libemq.a

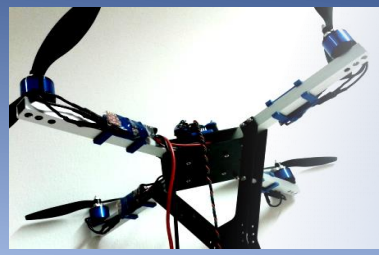


Aufgaben

Hilfe und Hintergrund:
Du kannst die Aufgaben prinzipiell entweder mit dem Projekt Framework oder *Framework_School* bearbeiten. Die Lösung findest du im Projekt *EMQ_QCS*. Verwende für den Kurs das Projekt *Framework_School*. Füge die Lösungen im Ordner *COM* in der Datei *communication.c* hinzu. Dazu dienen die folgenden zwei Funktionen:
Die Funktion `void my_communication_init()` wird einmalig aufgerufen, die Funktion `void my_communication()` wird wiederkehrend aufgerufen.

Aufgabe 1:
Sende den Text *„Hallo Welt“* über USART und lese die Nachricht über *HTerm* aus.
Verwende dazu die Funktionen `void my_communication_init()` und `void USART_write(char* line)`.

Aufgabe 2:
Lege in der Datei *communication.c* eine globale Zählvariable an. Sende diese mit der Funktion `void my_communication()` mit jedem Schleifendurchlauf an den PC und erhöhe sie anschließend. Verwende dazu die Funktionen `void sprintf(…)` und `void USART_write(…)`.



Aufgaben

Aufgabe 3:

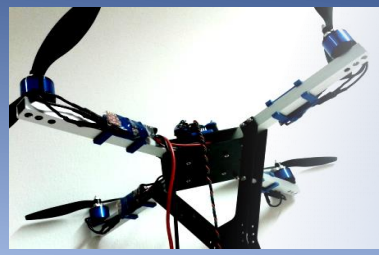
Programmiere die Echo-Funktion `void` **`my_echo_from_dma`**`()` in der Datei *communication.c ,* die eine Nachricht von *HTerm* einließt und unverändert wieder an den PC zurücksendet. Rufe diese Funktion von `void` **`my_communication`**`()` aus auf. Zum Auslesen kann die Funktion `dataStream` **`usart_read`**`()` verwendet werden. Speicher die gelesene Nachricht in der globalen Variable **`dataStream`** `my_data` zwischen. Vergiss nicht, am nach dem Auslesen den Übertragungskanal über `void` **`usart_release`**`()` wieder frei zu geben! **Hinweis:** Kommentiere die Befehle für das Senden des Counters aus Aufgabe 2 mit // vor der Code-Zeile aus.

Aufgabe 4:

Implementiere `void` **`my_usart_quadrierer`**`()`, die eine einstellige Ganzzahl über `dataStream` **`usart_read`**`()` einließt, mit sich selbst multipliziert und über USART wieder an den PC schickt. Nicht vergessen, den Übertragungskanal nach dem Lesen frei zu geben! **Hinweis:** Der Übertragungskanal wird nach `dataStream` **`usart_read`**`()` stets geleert!

Aufgabe 5:

Implementiere die Funktion `int` **`convert_chars_to_int`**`(`**`int`** `length,` **`char *`** `chars)`, die mehrere **`char`** Zeichen zum entsprechenden Wert als **`int`** zusammenfügt und zurückgibt. **Hinweis:** die Funktion **`pow10`**`(`**`double`** `x)` liefert das Ergebnis von 10 hoch x. Verwende die neue Funktion für die Lösung von Aufgabe 4 zum testen.



Aufgaben

Wichtige Hinweise:

- Mit `void` **`sprintf`**`(…)` lässt sich ein String füllen
- Ein String ist ein Array aus chars
- Ausreichend Speicher für String anlegen!
- Vorsicht: `void` **`sprint`**`(…)` prüft nicht, ob der Speicher reicht
- Zahl anlegen, die wir schreiben wollen
- String erstellen
- String versenden
- `'\n'` fügt einen Zeilenumbruch ein
- **`pow10`**`(`**`double`**` x)` liefert das Ergebnis von 10 hoch x

```
char my_string[80];

int zahl = 3;
sprintf(my_string, „Zahl ist %d", zahl);
USART_write(USART1, my_string);
```