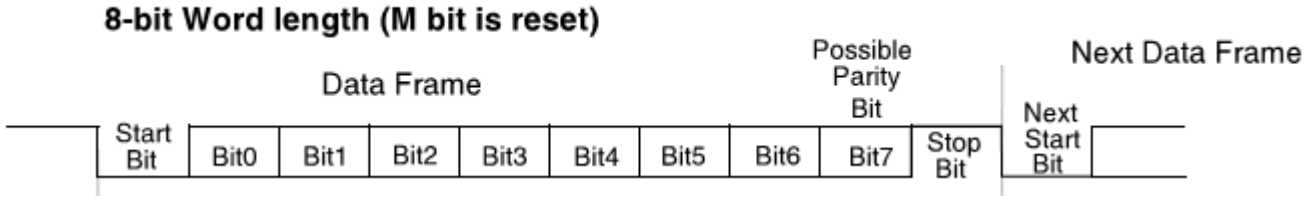
Versuch 2: Die serielle Schnittstelle (USART)

In diesem Termin wird die serielle Schnittstelle verwendet, um ASCII-Daten zwischen dem Mikrocontroller und einem Terminal (putty; Einstellungen siehe separates Dokument) zu übertragen. Diese Übertragung wird als V.24 bzw. RS232 Schnittstelle bezeichnet.

Kurze Einführung zur seriellen Schnittstelle:
Es werden 8 Datenbit sequentiell auf einer Leitung übertragen, wobei kein Takt übertragen wird. Sender und Empfänger müssen daher auf die gleiche Taktrate eingestellt werden. Damit der Beginn einer Datenübertragung vom Empfänger erkannt werden kann, werden die 8 Datenbit mit einem Start-Bit und einem Stop-Bit versehen (= Frame). Das Start-Bit hat einen Low-Pegel, das Stopp-Bit und der Idle-Zustand werden mit einem High-Pegel belegt. Damit entsteht beim Beginn einer Datenübertragung beim Übergang vom Idle-Zustand zum Start-Bit bzw. vom Stopp-Bit zum Start-Bit immer eine negative Flanke, die den Empfänger triggert. Das folgende Bild zeigt die Übertragung von 8 Bit ohne Parity-Bit.



Die Übertragungsgeschwindigkeit wird als Baud-Rate bezeichnet mit der Einheit Bit/sec. Eine typische Baudrate sind 9600 Baud d.h. 9600 Bit/sec, bei 10 Bit pro Zeichen dauert die Übertragung eines Zeichens dabei ca. 1 ms. Wenn also mehrere Zeichen hintereinander gesendet werden, darf erst ein neues Zeichen in das **Datenregister** (= Transmitregister) des USART-Bausteins geschrieben werden, wenn das Transmit-Register wieder leer ist. Der Baustein besitzt daher neben dem Datenregister ein **Statusregister**, in dem ein Bit (TXE = Transmitregister empty) gesetzt wird, sobald ein neuer Wert geschrieben werden kann.
Ein USART-Baustein kann senden und empfangen, ein weiteres Bit im Statusregister (RXNE = Receiveregister not empty) zeigt an, dass ein Datenpaket mit 8-Bit empfangen wurde und aus dem Datenregister gelesen werden kann.

Der Baustein hat weitere **Controlregister**, um Übertragungsmodi und die Baudrate einzustellen.

Die Register des USART-Bausteins im Überblick

Offset	Register	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x00	**USART_SR**	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NF	FE	PE
	Reset value																							0	0	1	1	0	0	0	0	0	0
0x04	**USART_DR**	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DR[8:0]								
	Reset value																								0	0	0	0	0	0	0	0	0
0x08	**USART_BRR**	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DIV_Mantissa[15:4]												DIV_Fraction [3:0]			
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x0C	**USART_CR1**	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	OVER8	Res.	UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
	Reset value																	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0
0x10	**USART_CR2**	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	LINEN	STOP [1:0]		CLKEN	CPOL	CPHA	LBCL	Res.	LBDIE	LBDL	Res.	ADD[3:0]			
	Reset value																		0	0	0	0	0	0	0	0	0	0		0	0	0	0
0x14	**USART_CR3**	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ONEBIT	CTSIE	CTSE	RTSE	DMAT	DMAR	SCEN	NACK	HDSEL	IRLP	IREN	EIE
	Reset value																					0	0	0	0	0	0	0	0	0	0	0	0
0x18	**USART_GTPR**	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	GT[7:0]								PSC[7:0]							
	Reset value																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Aufgabe 2.1: String ausgeben
Die USB-Schnittstelle ist fest mit USART2 verbunden. Schreiben Sie ein Programm, das einen String auf der Terminalemulation putty ausgibt. Vervollständigen Sie die Funktion **InitUSART** in `USART2.C`, die die entsprechenden Einstellungen (9600 Baud, 1 Stopp-Bit, kein Parity) durchführt. Schreiben Sie dann eine Funktion **WriteChar(char),** die ein Zeichen ausgibt. Beachten Sie dabei, dass Sie solange aktiv warten müssen (das TXE-Bit abfragen), bis das Transmitregister frei ist. Verwenden Sie anschließend diese Funktion, um die Funktion **WriteString(char *str)** zu realisieren, die einen kompletten String auf dem Terminal ausgibt.
Hinweis: Die Einstellung der Baudrate erfolgt im Register BRR nach der Formel:

$$\text{Tx/ Rx baud} = \frac{f_{CK}}{(16*\text{USARTDIV})}$$

Die Taktfrequenz ist f_{CK} = 16 MHz, berechnen Sie den Teiler USARTDIV für die Baudrate 9600. Der ganzzahlige Teiler wird im Register BRR in den Bits 4…15 angegeben, die Kommastellen umgerechnet in 1/16 wird in den Bits 0…3 angegeben.

Verwenden Sie das vorgegebene Projekt `V2.1_USART` und ergänzen Sie es entsprechend.

Hinweis:
Statt der eigenen Header-Datei `owniо.h` wird nun die Header-Datei `stm32f4xx.h` (`#include <stm32f4xx.h>`) verwendet. Diese und eine Reihe weiterer Header werden vom Keil-Entwicklungssystem mitgeliefert.

Aufgabe 2.2: Zeichen einlesen
Schreiben Sie nun eine Funktion **char ReadChar(void)**, die wartet, bis ein Zeichen eingegeben wurde. Testen Sie die Funktion, indem Sie das eingelesene Zeichen um 1 erhöhen und mit `WriteChar` zurückschreiben. Damit sollte am Terminal ein verfälschtes Echo entstehen. Wenn 1 gedrückt wird, wird 2 zurückgeschrieben.

Aufgabe 2.3: Interruptgesteuertes Echo
Im Folgenden wird nun die USART-Schnittstelle im Interruptmodus verwendet. Unser Programm wird daher in Hauptprogramm und Interrupt-Handler aufgeteilt. Zunächst wird nur das Hauptprogramm entwickelt.

Die USART-Schnittstelle kann so konfiguriert werden, dass nach jedem empfangenen Zeichen ein Interrupt ausgelöst wird. In der Interrupt-Service-Routine soll ein Zeichen eingelesen und als Echo (mittels der Funktion `WriteChar`) wieder ausgegeben werden.
Die Interrupt-Service-Routine muss folgenden Aufbau haben (vordefiniert in `USART2.C`):

```
void USART2_IRQHandler (void)
{
  ……
}
```

Der Name `USART2_IRQHandler` ist in der Vektortabelle im Modul `startup_stm32f446xx.s` bereits vordefiniert und wird vom Compiler entsprechend erkannt. Sehen Sie sich dazu die Vektortabelle an.

Teil 3: Interruptgesteuerte Eingabe

In Teil 3 soll nun ein String bis zu einem definierten Endezeichen (Punkt) per Interrupt eingelesen und in einem Puffer abgelegt werden. Der String kann als Befehl verstanden werden, der die Wartezeit und Laufrichtung eines (über den SysTick) vorgegebenen Lauflichts ändert. Es sollen zwei Befehle realisiert werden:

- wxxx: Wartezeit vor dem Weiterschalten des Lauflichts beträgt xxx Millisekunden
- r0 oder r1: Richtung des Lauflichts links oder rechts

Hinweis:

- In der Interruptroutine muss jedes eingegebene Zeichen nacheinander zu einem String aufgebaut werden. Es muss zusätzlich überprüft werden, ob das eingelesene Zeichen das Endezeichen ist.
- `sscanf` eignet sich bestens dazu, um Zahlen aus einem ASCII-String herauszulesen. Informieren Sie sich dazu über die Funktion `sscanf`!
- Um den String mit der Funktion `sscanf` aus der C-Standardlibrary analysieren zu können, muss er mit einer 0 abgeschlossen werden.
- siehe http://en.wikibooks.org/wiki/C_Programming/C_Reference/stdio.h/scanf