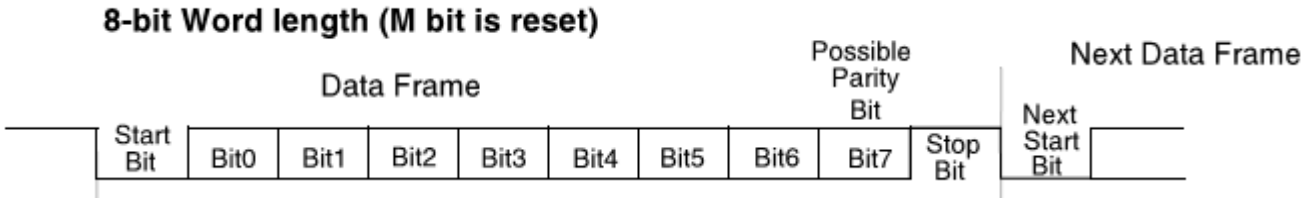
Versuch2: Die serielle Schnittstelle (USART)

In diesem Termin wird die serielle Schnittstelle verwendet, um ASCII-Daten zwischen dem Mikrocontroller und einem Terminal (putty) 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 den 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 Benennung Bit/sec. Eine typische Baudrate sind 9600 Baud d.h. 9600 Bit/sec, bei 10 Bit pro Zeichen dauert die Übertragung eines Zeichens 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 Datenregister gelesen werden kann.

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

Die Register des USART-Bausteins im Überblick

Table 130. USART register map and reset values

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**	Reserved																						CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NE	FE	PE
	Reset value																							0	0	1	1	0	0	0	0	0	0
0x04	**USART_DR**	Reserved																						DR[8:0]									
	Reset value																							0	0	0	0	0	0	0	0	0	0
0x08	**USART_BRR**	Reserved																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**	Reserved																		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
0x10	**USART_CR2**	Reserved																	LINEN	STOP [1:0]		CLKEN	CPOL	CPHA	LBCL	Reserved	LBDIE	LBDL	Reserved	ADD[3:0]			
	Reset value																		0	0	0	0	0	0	0		0	0		0	0	0	0
0x14	**USART_CR3**	Reserved																					CTSIE	CTSE	RTSE	DMAT	DMAR	SCEN	NACK	HDSEL	IRLP	IREN	EIE
	Reset value																						0	0	0	0	0	0	0	0	0	0	0
0x18	**USART_GTPR**	Reserved																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

Schreiben Sie ein Programm, das einen String auf der Terminalemulation putty ausgibt. Schreiben Sie eine Funktion **InitUSART**, 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 f_{CK} = 24 MHz, berechnen Sie den Teiler USARTDIV für die Baudrate 9600. Der ganzahlige Teiler wird im Register BRR von Bit 4..15 angegeben, die Kommastellen umgerechnet in 1/16 wird von Bit 0..3 angegeben.

Verwenden Sie das vorgegebene Projekt V2_1_USART_out und ergänzen Sie es entsprechend.

Hinweis:

Statt der eigenen Header-Datei „ownio.h“ wird nun die Header-Datei stm32f10x.h (#include <stm32f10x.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, in dem 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: Zeichen per Interrupt einlesen
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. Verwenden Sie dazu die Vorlage V2.3_USART_DAC.

Teil1: Sinusausgabe
Im Hauptprogramm werden in einer Endlosschleife mit Hilfe des Digital-Analogwandlers (DAC) die Werte einer Sinuskurve ausgegeben, die am Oszilloskop sichtbar gemacht werden. Die Sinuskurve wird in 32 Schritten in einem maximalen Wertebereich von 0 bis 4095 dargestellt. Zur Ausgabe der Analogwerte sollen die vorgegebenen Funktionen InitDAC und WriteDAC() verwendet werden.
Schreiben Sie das Programm zur Sinusausgabe, am Oszi sollte dann eine stufige Sinuskurve sichtbar werden..

Wir verwenden einen 12-Bit DA-Converter, d.h. es können Werte von 0 bis 4095 verwendet werden. Nachdem der DA-Converter des STM32 in den Randbereichen unsauber arbeitet, soll der Bereich nur zu ca. 90% ausgenutzt werden (2048 +/- 1850)
Bereiten Sie eine Tabelle mit 32 Sinuswerten vor.

Teil2. Interruptgesteuertes Echo
Die USART-Schnittstelle kann so konfiguriert werden, dass nach jedem empfangenen Zeichen ein Interrupt ausgelöst wird. In der InterruptServiceRoutine soll ein Zeichen eingelesen und als Echo (mittels der Funktion WriteChar) wieder ausgegeben werden.
Die Interruptservice Routine muss folgenden Aufbau haben:

```
void USART1_IRQHandler (void)
{
  ……
}
```

Der Name USART1_IRQHandlerist in der Vektortabelle im Modul „startup_stm32f10x_md.s“ bereits vordefiniert und wird vom Compiler entsprechend erkannt. Sehen Sie sich dazu die Vektortabelle im Modul „startup_stm32f10x_md.s“ an.

Teil3. Interruptgesteuerte Eingabe

In Teil3 soll nun ein String bis zu einem definierten Endezeichen (Punkt als Endezeichen) per Interrupt eingelesen und in einem Puffer abgelegt werden. Der String kann als Befehl verstanden werden, der den Wert der Sinustabelle an einer bestimmten Position ändert. wird, während im Hauptprogramm weiterhin das Sinussignal ausgegeben wird.
Der eingelesene String gibt eine Position (0..31) und einen Wert (0..4095) an und hat das Format „ p12w1250.“ (Position 12 und Wert 1250, der Punkt stellt das Endezeichen dar).

Analysieren Sie in Ihrem Programm den String und setzen dann in der Sinustabelle an der angegebenen Position den neuen Wert. Auf dem Oszi sollte dann der neue Wert sichtbar sein.

Hinweis:

- In der Interruptroutine muss jedes eingegebene Zeichen nacheinander in einem String abgelegt 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