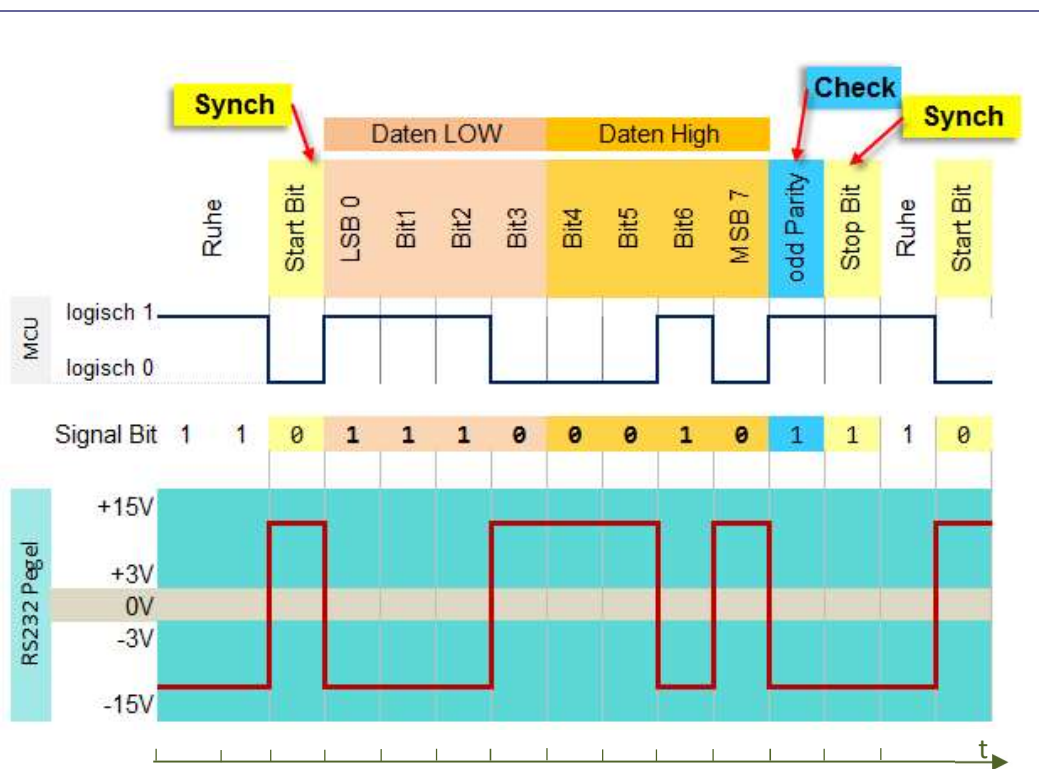


Fach HST

Datenübertragung

Serielle Übertragung von Daten



Version: 2143.00

1 Inhaltsverzeichnis

1	Inhaltsverzeichnis.....	2
2	Serielle Datenübertragung RS232	3
2.1	Stecker 9 Polig	3
2.2	Stecker aus Sicht eines PC's	3
2.2.1	Hardware-Datenflusskontrolle.....	3
2.2.2	Software- Datenflusskontrolle.....	3
2.3	Elektrische Eigenschaften / Pegel [2]	4
2.4	Datenübertragungsbeispiel	5
3	Die serielle Schnittstellen auf dem MCB32: System	6
3.1	System Architektur	6
3.2	Betriebsarten STM32F107VC MCU.....	6
4	Die serielle Schnittstelle MCB32: Details	7
4.1	Betriebsarten	7
4.2	Blockdiagramm USART	7
4.3	Baudratengenerierung	8
4.3.1	Auszug aus dem Referenz Manual für USART_BRR.....	8
4.4	Programmierung der seriellen Schnittstelle USART2.....	9
4.4.1	Referenz Manual: APB2 peripheral clock enable register (RCC_APB2ENR) (Zeile 4:).....	9
4.4.2	Referenz Manual: 8.3.8 APB1 peripheral clock enable register (RCC_APB1ENR) (Zeile 5:).....	10
4.4.3	Referenz Manual: Port configuration register low (GPIOx_CRL) (x=A..G) (Zeile 6 & 7)	10
4.4.4	Referenz Manual: AF remap and debug I/O configuration register (AFIO_MAPR) (Zeile 8).....	10
4.4.5	Referenz Manual: Control register 1 (USART_CR1) (Zeile 10 und 11)	11
4.4.6	Referenz Manual: Status register (USART_SR) (Zeile 12 und 15).....	11
5	Programmbeispiele und Aufgaben.....	12
5.1	Beispiel 1: Der µC sendet Zeichen an den PC	12
5.2	Beispiel 2: Der PC sendet Zeichen an den µC	13
6	Aufgaben SERIAL Programmierung.....	14
7	Hausaufgabe/ Übung: Serielle Schnittstelle	15

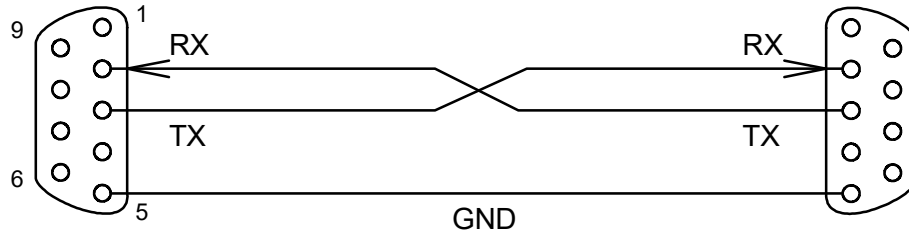
Appendix

A1	Ascii Tabelle.....	18
A2	Abkürzungen.....	19
A3	Anhang Referenzen	20
A4	Programmierung der seriellen Schnittstelle USART2 mit der Touch-Library.....	21
A4.1	Auszug aus TouchP0P1 Library Befehlsliste	22
A5	Auftrag: Serielle Schnittstelle	23
A7	Lösungen Serial 1 ... Serial 4.....	25

2 Serielle Datenübertragung RS232

2.1 Stecker 9 Polig

Schnittstellenanschluss am 9-poligen D-Sub-Stecker Minimalkonfiguration



Aus Sicht des OSI-Schichtenmodells ist die Schnittstelle dem Layer 1 (Physical Layer) zuzuordnen.

2.2 Stecker aus Sicht eines PC's

Die folgende Tabelle zeigt die Anschlussbeschreibung aus Sicht eines PC's. Der Stecker mit seinen 9 Anschlüssen ist wie folgt definiert

Pin	Beschreibung	Abkürzung	Richtung (vom PC aus gesehen)	Bemerkung
1	carrier detect	CD	in	
2	receive data	RxD	in	Daten zum PC
3	send data	TxD	out	Daten vom PC
4	data terminal ready	DTR	out	PC bereit
5	ground		GND	
6	data set ready	DSR	in	Mikrocontroller bereit
7	request to send	RTS	out	PC möchte Daten senden
8	clear to send	CTS	in	Mikrocontroller bereit zum Daten empfangen
9	ring indicate	RI	in	Eingehender Anruf (bei Modems)

2.2.1 Hardware-Datenflusskontrolle

Neben dem Empfangen und Senden von Daten gibt es noch Anschlüsse für die Datenflusskontrolle. Sie wird benötigt um den Datenfluss zu steuern. Das wird dann nötig wenn ein Gerät die Daten schneller liefert als das Andere es empfangen kann.

- Vorteil: Transparente Datenübertragung, d.h. es müssen keine Steuerbytes übertragen werden
- Nachteil: Es werden zwei zusätzliche Leitungen benötigt

Die Empfangsbereitschaft des PCs wird gemäss obiger Tabelle über das Signal DTR angezeigt.

2.2.2 Software- Datenflusskontrolle

Die Datenflusskontrolle findet über zwei «**Steuerzeichen**» statt:

XOFF (0x13) stoppt den Datenfluss

XON (0x11) setzt Datenübertragung fort.

Wenn z.B. der PC ein XOFF sendet, bedeutet dies für den Mikrocontroller, dass er keine Daten mehr an den PC senden darf. Diese beiden Zeichen dürfen aber im normalen Datenverkehr nicht vorkommen! [1]

2.3 Elektrische Eigenschaften / Pegel [2]

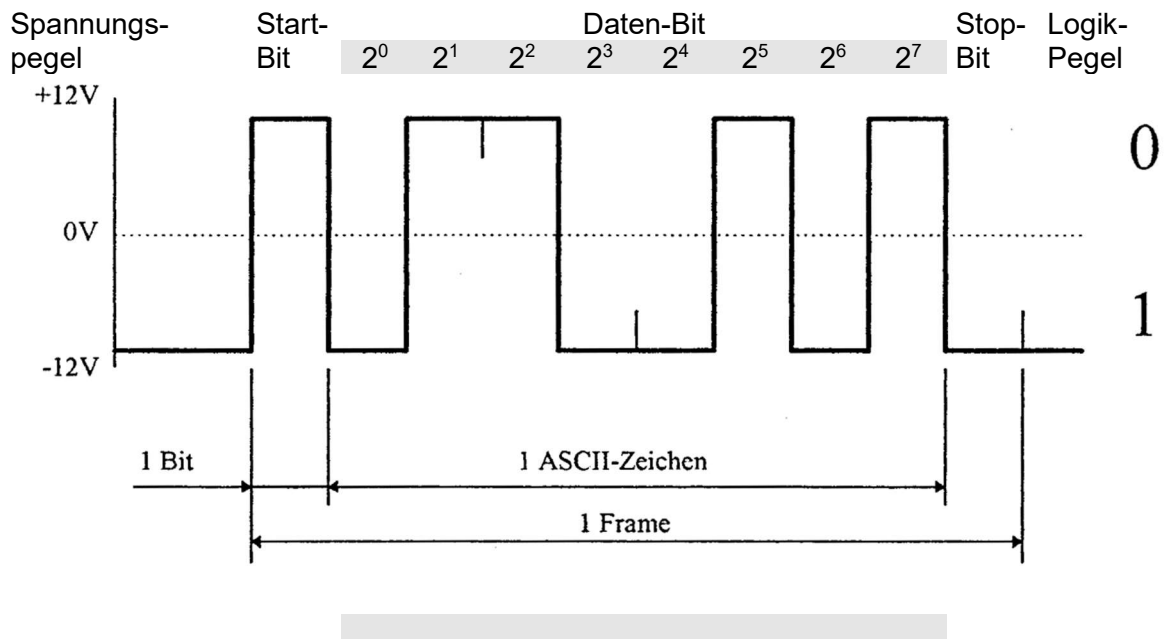
Achtung. Die Signalspannung darf nie zwischen +3 -3V zu liegen kommen.

Logik Pegel uP	Signalpegel auf der RS232 Lei- tung	Spannung auf der Datenleitung	Spannung auf den Steuer- und Meldeleitungen
1	L (low)	-3V -15V	+3V +15V
0	H (high)	+3V +15V	-3V -15V

Bei RS232 Verbindungen ist die maximale Kabellänge vom Kabeltyp, von der Übertragungsrate sowie der Signalstärke abhängig. Je höher die Rate desto kürzer das Kabel. Im Normalfall ist ein Kabel zw. 6 – 8 m lang. Bei optimalen Voraussetzungen kann es auch bis zu 40m sein.

2.4 Datenübertragungsbeispiel

Man beachte im folgenden Beispiel die Spannungspegel. Sie bewegen sich zwischen -15 und +15Volt, im folgenden Beispiel +/-12V.



Aufgabe: Welches ASCII-Zeichen ist im obigen Signalverlauf dargestellt?

Lösung:

Wertigkeit:	2 ⁰	2 ¹	2 ²	2 ³	2 ⁴	2 ⁵	2 ⁶	2 ⁷
Wert:								

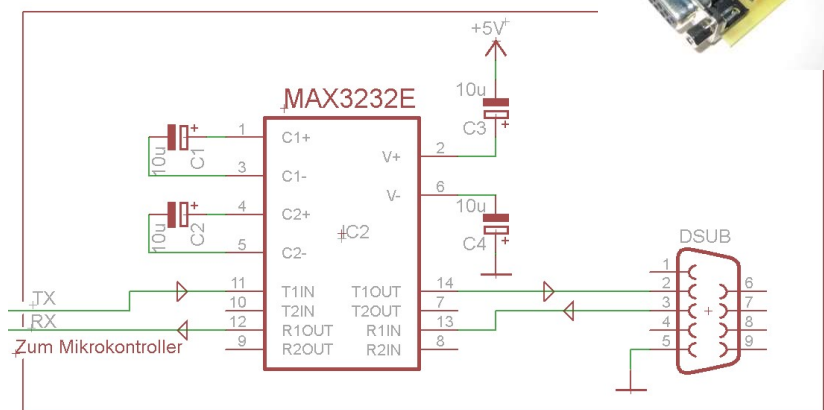
Übertragungsgeschwindigkeit: Baudrate = Anzahl Bit pro Sekunde

bei 9600 Baud:

T_{Bit}
T_{Frame}

Die hohen Spannungspegel kommen nicht aus dem Controller sondern sie werden von speziellen Modulen / Chips erzeugt welche die TTL-Pegel der Controller in die RS232-Pegel umwandeln. (Beide Richtungen)

Das Schema zeigt eine mögliche Schaltung mit einem MAX232 Chip. Der Chip kann entweder 2 Kanäle verarbeiten oder die Hardwaresignale für die Datenflusskontrolle liefern. ((DSR an Pin 7 und RTS an Pin 8 usw).



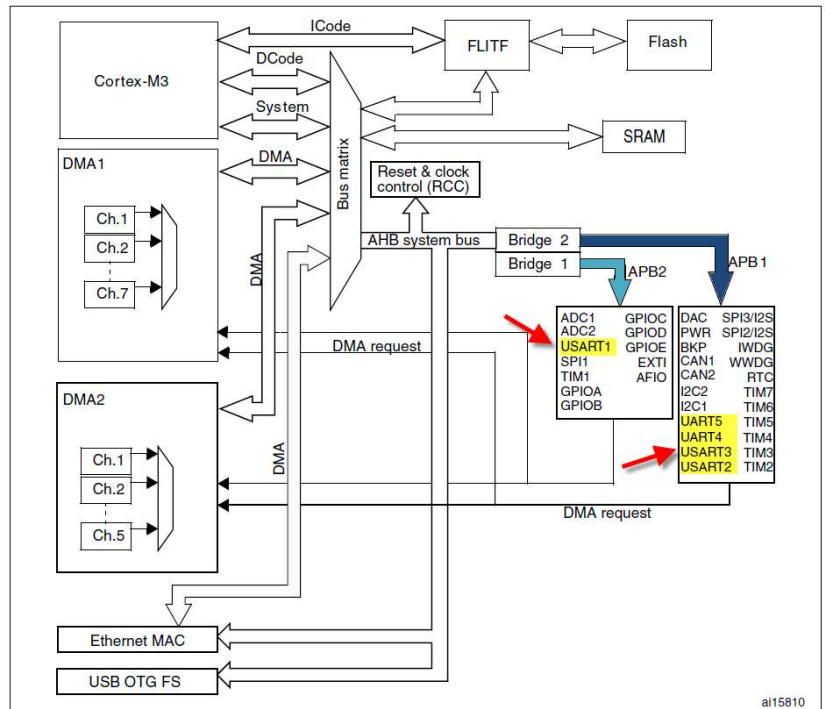
3 Die serielle Schnittstellen auf dem MCB32: System

3.1 System Architektur

Zu beachten ist, dass die USARTs verschiedene Quellen betreffend Ansteuerung haben. Einerseits den APB1 (APB-Advanced Peripheral Bus) und den APB2. Der APB2 kann mit bis zu 72MHz arbeiten und der APB1 nur bis zu 36MHz. Dieser Umstand muss bei der Planung der Schnittstellen und auch bei der Programmierung beachtet werden:

USART1 via **APB2** (72MHz)
und

USART2 via **APB1** (36MHz)



3.2 Betriebsarten STM32F107VC MCU

Die seriellen Schnittstellen haben folgende Eigenschaften:

Table 199. USART mode configuration⁽¹⁾

USART modes	USART1	USART2	USART3	UART4	UART5
Asynchronous mode	X	X	X	X	X
Hardware Flow Control	X	X	X	NA	NA
Multibuffer Communication (DMA)	X	X	X	X	NA
Multiprocessor Communication	X	X	X	X	X
Synchronous	X	X	X	NA	NA
Smartcard	X	X	X	NA	NA
Half-Duplex (Single-Wire mode)	X	X	X	X	X
IrDA	X	X	X	X	X
LIN	X	X	X	X	X

1. X = supported; NA = not applicable.

Die 3 USART's besitzen alle Möglichkeiten um programmiert werden zu können. Also Hardware-Flusssteuerung, asynchronen und synchronen Betrieb usw.

Wir werden bei unseren Anwendungen USART1 und 2 benutzen weil diese als echte RS232 Schnittstelle ausgeführt sind.

4 Die serielle Schnittstelle MCB32: Details

4.1 Betriebsarten

Das MCB32 Board stellt 2 vollwertige RS232-Kanäle zur Verfügung. Das bedeutet die Pegel sind im Bereich von +/-15V und es muss entsprechend vorsichtig gearbeitet werden. Grundsätzlich können die Signale auch direkt an den Ports des STM32F107VC abgenommen werden. In unserem Fall sind das die Ports PB(6/7) und PD(5/6). Pegel von 0/3.3V.

4.2 Blockdiagramm USART

USART1 und USART2 sind auf dem MCB32-Board durch Remap auf Alternativpins umgeleitet; siehe AFIO_MAPR. Das folgende Bild zeigt USART2 (max. 36MHz)

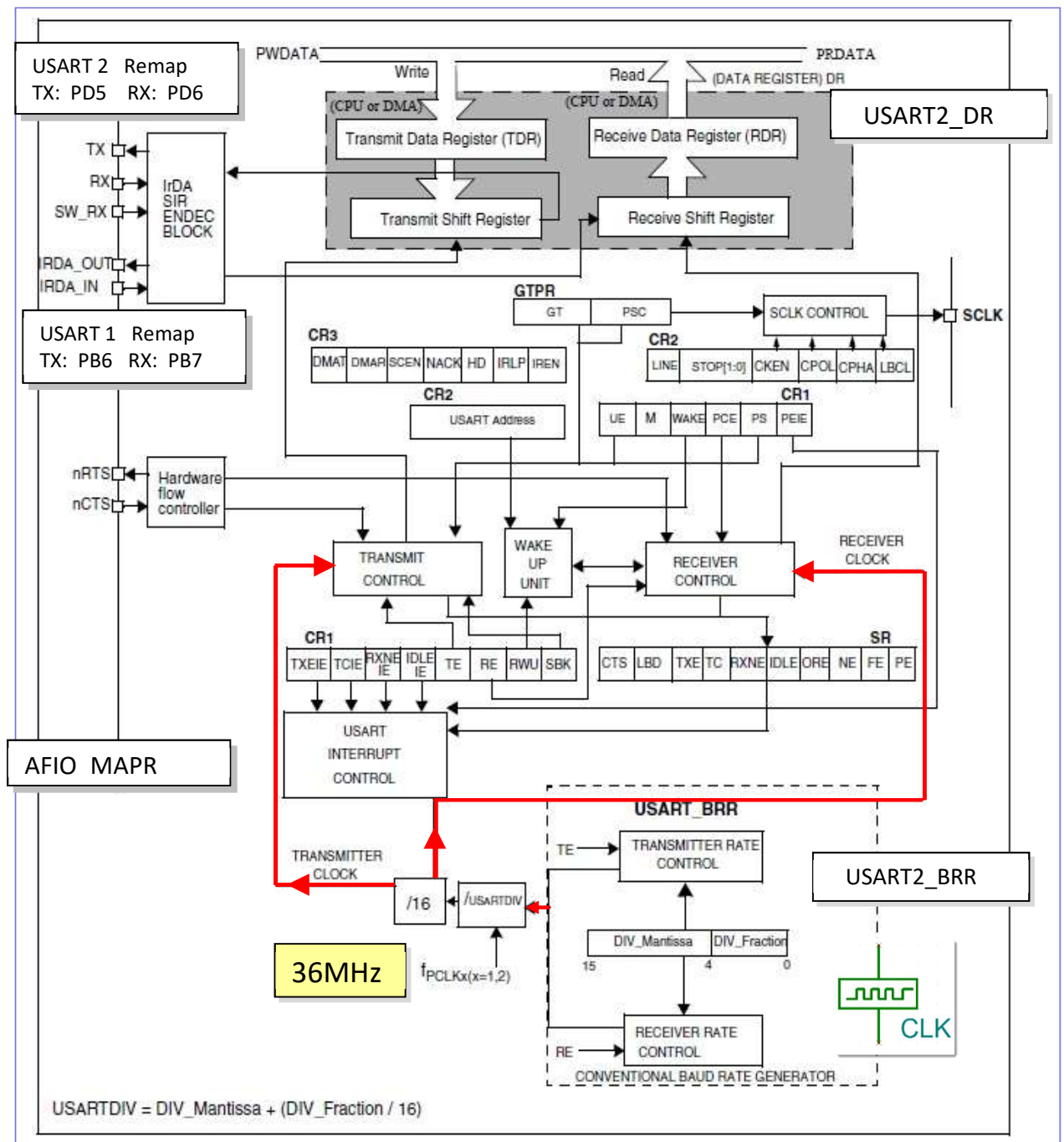


Abbildung 1: USART Blockdiagramm

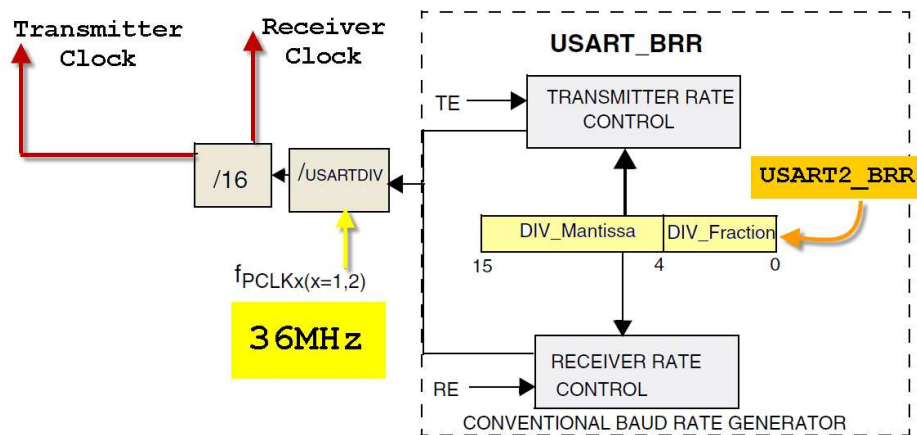
4.3 Baudratengenerierung

Die Baudrate wird mit einem separaten Baudratengenerator USART_BRR erzeugt. Der Timer wird mit der halben Oszillator-Frequenz getaktet.

$$T_X/R_X \text{ Baudrate} = \frac{f_{ck}}{16 * USARTDIV} \quad [4-1]$$

USARTDIV ist der Teilerfaktor. Dieser Wert wird aufgeteilt und der Ganzzahl sowie der Dezimalzahl-Teil im Register **DIV_Mantisse** und **DIV_Fraction** gespeichert.

Beispiel: USART2 (36MHz), 9600Baud: ergibt einen USARTDIV von 234.375. Nun wird 234 in HEX → 0xEA als **DIV_Mantisse** und der Dezimalteil 0.375 mit 16 multipliziert = 0x6 in **DIV_Fraction** umgesetzt. **DIV_Mantisse** muss noch richtig platziert werden. Konkret wird das Register BRR mit diesem Wert geladen. Man kann dies für den des USART2 Kanal auch direkt codieren:
USART_BRR2 = DIV_Mantisse_{Hex} <<4 + (16* DIV_Fraction)_{Hex}



Aufgabe: Der Schnittstelle 2 soll als 8-Bit-UART mit einer Baudrate von 19200 Bd codiert werden. Welcher HEX-Wert muss in das Register USART_BRR2 geladen werden?

Lösung:

4.3.1 Auszug aus dem Referenz Manual für USART_BRR

27.6.3 Baud rate register (USART_BRR)

Note: The baud counters stop counting if the TE or RE bits are disabled respectively.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIV_Mantissa[11:0]												DIV_Fraction[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

4.4 Programmierung der seriellen Schnittstelle USART2

Die Programmierung erfolgt bei Benutzung der hardwarenahen Programmierung über mehrere Register (Siehe Referenz Manual). Die Bedeutung der einzelnen Bits wird im nachfolgenden Beispiel erläutert.

```
// Serielles Einlesen eines Buchstabens
// an USART2 PD6 und 1 erhöht Ausgabe an PD5.
// Per Terminal am PC via Hypterterminal einlesen
//-----
#include <stm32f10x.h> //1:uC Mikrokontrollertyp
#include "TouchP0P1.h" // P0/P1,8Bit,Touchscreen/ Grafik

int main(void) // 2:Hauptprogramm
{
    char c = '0';

    RCC->APB2ENR |= (1<<5)|1; // 3:USART2 initialisieren:
    RCC->APB1ENR |= 1<<17; // 4:PortD + AFIO Enable
    GPIOD->CRL &= 0xF0FFFFFF; // 5:USART2 Clock ON 36MHz
    GPIOD->CRL |= 0x04B00000; // 6:PD5 AF Out 50MHz und ..
    AFIO->MAPR |= 1<<3; // 7:PD6 In Floating
    USART2->BRR = 0xEA6; // 8:USART2 Remap TX-PD5 RX-PD6
    USART2->CR1 |= 3<<2; // 9:9600Bd @ 36MHz
    USART2->CR1 |= 1<<13; //10:TX und RX Enable
    //11:USART Enable

    while(1) // Endlosschlaufe
    {
        if(USART2->SR&(1<<5)) //12:Zeichen empfangen
        { //13:Zeichen lesen von RX
            c = USART2->DR;
            USART2->DR = c+1; //14:Zeichen +1 an TX
            while(!(USART2->SR&(1<<7))); //15:Warte bis TX frei
        }
    } // while(1)
} // Main
```

4.4.1 Referenz Manual: APB2 peripheral clock enable register (RCC_APB2ENR) (Zeile 4:)

In der Zeile // 4: wird mit dem Code: `RCC->APB2ENR |= (1<<5)|1;` das Bit 0 und das Bit 5 im Register RCC_APB2ENR auf 1 gesetzt. Die Bedeutung der Bits ist im Auszug des Ref. Manuals sofort klar.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	USART 1EN	Res.	SPI1 EN	TIM1 EN	ADC2 EN	ADC1 EN	Reserved	IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.	AFIO EN	
	rw		rw	rw	rw	rw		rw	rw	rw	rw	rw		rw	

Bit 5 IOPDEN: I/O port D clock enable

Set and cleared by software.
0: I/O port D clock disabled
1: I/O port D clock enabled

Bit 0 AFIOEN: Alternate function I/O clock enable

Set and cleared by software.
0: Alternate Function I/O clock disabled
1: Alternate Function I/O clock enabled

4.4.2 Referenz Manual: 8.3.8 APB1 peripheral clock enable register (RCC_APB1ENR) (Zeile 5:)

In der Zeile // 5: wird mit dem Code: `RCC->APB1ENR |= 1<<17;` das Bit 17 im Register RCC_APB1ENR auf 1 gesetzt. Damit wird der USART2 eingeschaltet.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		DAC EN	PWR EN	BKP EN	CAN2 EN	CAN1 EN	Reserved		I2C2 EN	I2C1 EN	UART5 EN	UART4 EN	USART3 EN	USART2 EN	Res.
		r/w	r/w	r/w	r/w	r/w			r/w	r/w	r/w	r/w	r/w	r/w	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 EN	SPI2 EN	Reserved		WWD GEN	Reserved					TIM7 EN	TIM6 EN	TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN

Bit 17 USART2EN: USART 2 clock enable

Set and cleared by software.

0: USART 2 clock disabled

1: USART 2 clock enabled

4.4.3 Referenz Manual: Port configuration register low (GPIOx_CRL) (x=A..G) (Zeile 6 & 7)

Mit der Befehlssequenz: `GPIOD->CRL &= 0xF00FFFFF;` und `GPIOD->CRL |= 0x04B00000;` wird der Port D mit seinen beiden Ports PD5 und PD6 richtig für die Anwendung als USART-Pins konfiguriert. Die erste Sequenz setzt die Pins zurück (Reset State) und mit der zweiten Sequenz wird PD6 als Input floatend gesetzt und PD5 in dem Mode Output, Push-Pull AF (AlternateFunction) mit 50MHz Speed gesetzt.

31	30	29	28	Bit PD6				Bit PD5				19	18	17	16
CNF7[1:0]		MODE7[1:0]		CNF6[1:0]		MODE6[1:0]		CNF5[1:0]		MODE5[1:0]		CNF4[1:0]		MODE4[1:0]	
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNF3[1:0]		MODE3[1:0]		CNF2[1:0]		MODE2[1:0]		CNF1[1:0]		MODE1[1:0]		CNF0[1:0]		MODE0[1:0]	
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

MODEy[1:0]: Port x mode bits (y= 0 .. 7)

00: Input mode (reset state)
 01: Output mode, max speed 10 MHz.
 10: Output mode, max speed 2 MHz.
 11: Output mode, max speed 50 MHz.

CNFy[1:0]: Port x configuration bits (y= 0 .. 7)

In input mode (**MODE[1:0]=00**):

00: Analog mode
 01: Floating input (reset state)
 10: Input with pull-up / pull-down
 11: Reserved

In output mode (**MODE[1:0] > 00**):

00: General purpose output push-pull
 01: General purpose output Open-drain
 10: Alternate function output Push-pull
 11: Alternate function output Open-drain

Bemerkung: Alternative Funktion Open-Drain und Alternative Funktion Push-Pull müssen gewählt werden, falls der Pin als Ausgang eines Moduls wie etwas USART, SPI, I22, etc. genutzt wird

4.4.4 Referenz Manual: AF remap and debug I/O configuration register (AFIO_MAPR) (Zeile 8)

Wir wollen nun dem USART2 die Pins PD5 und PD6 zuordnen. Dies geschieht über die Remapping –Funktion mit dem Befehl: `AFIO->MAPR |= 1<<3;`

Alternate functions	USART2_REMAP = 0	USART2_REMAP = 1 ⁽¹⁾
USART2_CTS	PA0	PD3
USART2_RTS	PA1	PD4
USART2_TX	PA2	PD5
USART2_RX	PA3	PD6
USART2_CK	PA4	PD7

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PD01_REMAP	CAN_REMAP [1:0]		TIM4_REMAP	TIM3_REMAP [1:0]		TIM2_REMAP [1:0]		TIM1_REMAP [1:0]		USART3_REMAP[1:0]		USART2_REMAP	USART1_REMAP	I2C1_REMAP	SPI1_REMAP
r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w	r/w

Bit 3 USART2_REMAP: USART2 remapping

This bit is set and cleared by software. It controls the mapping of USART2 CTS, RTS, CK, TX and RX alternate functions on the GPIO ports.

0: No remap (CTS/PA0, RTS/PA1, TX/PA2, RX/PA3, CK/PA4)

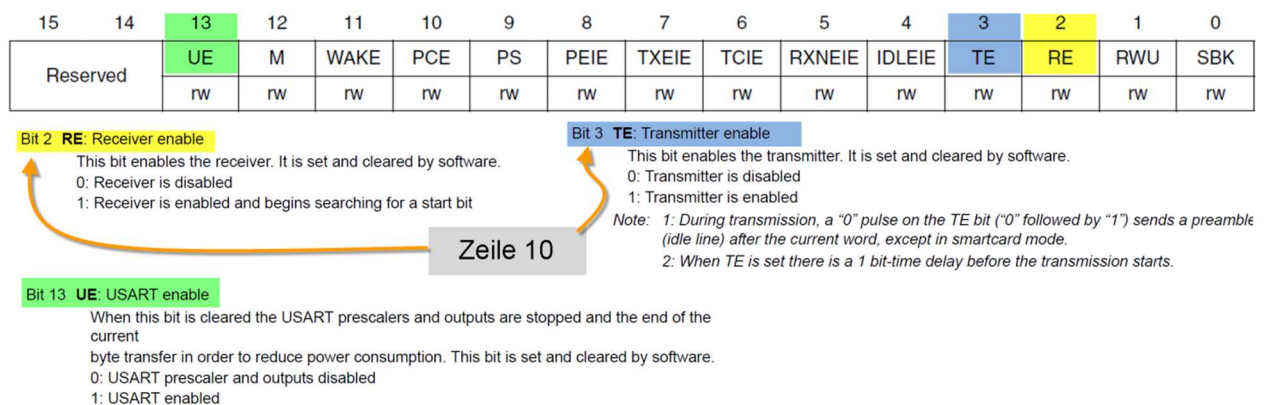
1: Remap (CTS/PD3, RTS/PD4, TX/PD5, RX/PD6, CK/PD7)

Als nächsten Schritt programmieren wir die Baudrate von USART2 mit 9600Baud.

`USART2->BRR = 0xEA6;` Die Berechnung des Wertes 0xEA6 ist beschrieben, siehe 4.3.

4.4.5 Referenz Manual: Control register 1 (USART_CR1) (Zeile 10 und 11)

Zum Schluss wird nun der USART eingeschaltet. Das geschieht in 2 Schritten. Zuerst werden die Pins eingeschaltet `USART2->CR1 |= 3<<2;` und dann der gesamte USART `USART2->CR1 |= 1<<13;`.



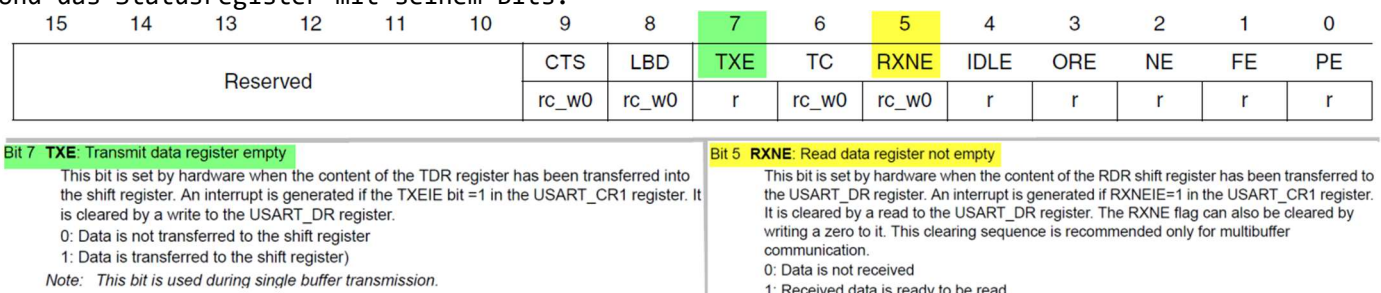
So. Nun läuft der USART Kanal2 und wir können Daten empfangen und versenden. Dies passiert in den Zeilen 12 / 13 und 14. In der Zeile 12 werten wir ein Bit aus. Dieses Bit zeigt ob der Empfangsbuffer ein Zeichen empfangen hat. In der Zeile 15 machen wir das Gleiche mit den Senderbuffer. Auch dies ist eine Art der Datenflusskontrolle.

4.4.6 Referenz Manual: Status register (USART_SR) (Zeile 12 und 15)

Codezeilen welche Daten empfangen und senden:

```
while(1) // Endlosschleife
{
    if(USART2->SR&(1<<5)) //12:Zeichen empfangen
    { //13:Zeichen lesen von RX
        c = USART2->DR;
        USART2->DR = c+1; //14:Zeichen +1 an TX
        while(!(USART2->SR&(1<<7))); //15:Warte bis TX frei
    }
}
```

Und das Statusregister mit seinem Bits.



5 Programmbeispiele und Aufgaben

```
//-----  
// Titel      : Datentransfer MCB32 -> PC  
// Datei      : SER_BSP1.C  
// Erstellt   : 12.2.2016 / rma  
// Funktion   : Die Zeichen von 'A' .. 'z' werden von der USART2-  
//              Schnittstelle des MCB32 an den PC gesendet.  
//-----  
  
// ---- Includes  
#include <.....>  
  
void InitUsart2(void) // USART2 mit 9600 Baud initialisieren  
{  
    // Code einfügen  
    //  
}  
  
void ZeichenSenden (char ch)  
{  
    // Code einfügen  
    // Ausgabepuffer <-- Zeichen  
    // warten bis Zeichen gesendet  
    // ev. Flags löschen löschen  
}  
  
void main ()  
{  
    char Zeichen;  
    InitUsart2();  
    for (Zeichen = 'A'; Zeichen <= 'z'; Zeichen++)  
        ZeichenSenden(Zeichen);  
    while (1);  
}
```

5.1 Beispiel 1: Der μ C sendet Zeichen an den PC

Aufgabe: Das Beispiel SER_BSP1.C ist einzugeben, fertig zu stellen und zu testen.

5.2 Beispiel 2: Der PC sendet Zeichen an den μC

```
/** @file SER_BSP2.C
 * @brief Serielles Einlesen eines Buchstabens vom Keyboard des PC
 * an USART2 PD6 und Ausgabe an PD5 via Terminal am PC
 * USART Init via spezielle Funktionen der TouchP0P1-Library
 * @author rma / TBZ
 */
//=====Inclu-
des=====
#include <stm32f10x.h> // Mikrocontrollertyp
#include "TouchP0P1.h" // P0/P1,8Bit,Touchscreen und Grafik

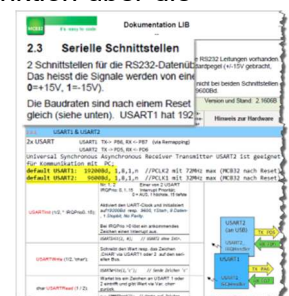
//=====main()=====

int main(void) // Hauptprogramm
{
    char c = '0'; // Starte mit Zeichen 0
    USARTInit(2, 0); // Default: USART2 ohne Intr. mit 9600Bd, 1,8,1,n
    InitTouchP0P1 ("1");

    while(1) // Endlosschleife
    {
        if(USARTtoRead(2)) // 1= Zeichen empfangen, 0 = kein Zeichen -
        {
            c = USARTRead(2); // Zeichen lesen von RX
            P1=c; // Zeichen an P1 ausgeben
            USARTWrite(2,c); // Sende Zeichen c
        }
    }
}
```

- Aufgabe:**
- a) Das Beispiel `SER_BSP2.C` ist einzugeben und zu testen.
 - b) Beispiel 2 soll so angepasst werden, dass dieselbe Funktion über die Schnittstelle USART1 realisiert werden kann.

Studiere dazu das Manual [3] mit allen Befehlen der Library oder dieses Skript (Seite 21 / A4).
Grund: USART1 Baudrate hat eine höhere Clockfrequenz f_{ck} .



Die Aufgabe ist als `SER_BSP3.C` zu codieren und zu testen.

6 Aufgaben SERIAL Programmierung

SERIAL1 Das MCB32 soll eine Bereitschaftsanzeige 'Ready' an das Terminal senden.

SERIAL2 An das Terminal ist in einer Endlosschleife ein ASCII-Zeichen zu senden, das den Schalterstellungen von Port P0 im HEX-Code entspricht. Das Programm ist so zu codieren, dass jedes Zeichen nach **einer Änderung** nur einmal gesendet wird!

SERIAL3 Ein Zeichen soll an das MCB32 gesendet, dort um den Wert 1 erhöht und an das Terminal zurückgesendet werden.

Programmstruktur

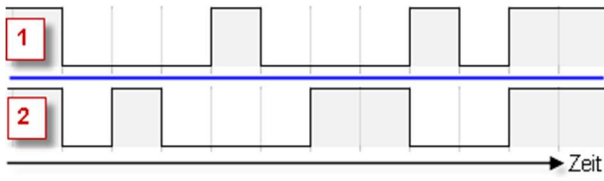
UP: InitUsart2 ()
UP: ZeichenEmpfangen ()
UP: ZeichenSenden ()
main
InitUsart2 ()
Wiederhole endlos
ZeichenEmpfangen ()
ZeichenSenden ()
ende

SERIAL4 Der Wert des Potentiometers auf dem MCB32 soll via einen Analog-Digitalwandler (ADC 1) soll im HEX-Code an das Terminal gesendet werden. Die Messungen sollen beliebig gemittelt werden können.

Programmstruktur

UP: InitADC ()	// ADC initialisieren
UP: char ADC_TP(nmal)	// nmal messen und Resultat gemittelt retour
UP: InitUsart2 ()	// COM initialisieren
UP: ZeichenSenden ()	// Zeichen an PC-COMxx senden
UP: Dez_Hex ()	// Dezial-Hex-Wandlung
main	
InitADC ()	
InitUsart2 ()	
Wiederhole endlos	
J	Wert geändert N
ZeichenSenden (Dez_Hex(Sechzehner))	
ZeichenSenden (Dez_Hex(Einer))	
ZeichenSenden (Space)	
ende	

7 Hausaufgabe/ Übung: Serielle Schnittstelle

	Frage	Lösung	
1.	<p>Zeichne den Signalverlauf an einer RS232 Schnittstelle für das Zeichen „A“.</p> <p>Wie lange dauert die Datenübertragung?</p> <p>Die Schnittstelle läuft mit 9600Baud, 1 Startbit, 1 Stopbit, oddParity.</p>		
2.	<p>Wieviel Start und Start-Bits kann ein Zeichen in einer seriellen Datenübertragung haben.</p> <p>Kann die Länge des Symbolen auch variieren. Wenn Ja in welchem Bereich.</p>	[4][5]	
3.	<p>Eine Datenübertragung mit 19200Baud sendet das Zeichen H und dann sofort das Zeichen 3. Wie lange dauert die Übertragung.</p> <p>Die Schnittstelle läuft mit 19200Baud, 1 Startbit, 1 Stopbit, No Parity.</p>		
4.	<p>Welche Zeichen werden mit folgender Sequenz (1 dann 2) übertragen. (TTL-Pegel)</p>  <p>Die Schnittstelle läuft mit TTL, xxBaud, 1 Startbit, 1 Stopbit, No Parity.</p>		
5.	<p>Die Übertragung des ersten Zeichens in obiger Aufgabe (4) beträgt 4.16ms. Wie gross ist die Baudrate.</p>		

	Frage	Lösung																														
6.	Wieviel Datenbits können pro Sekunde bei den angegebenen Baudraten übertragen werden wenn das Symbol mit 8Bit kodiert wird und je 1 Start- und Stopbit dazu gefügt wird.	<table><tr><th>Baudrate</th><th>Anzahl Datenbits</th></tr><tr><td>4800</td><td></td></tr><tr><td>9600</td><td></td></tr><tr><td>38'400</td><td></td></tr><tr><td>230'400</td><td></td></tr></table>		Baudrate	Anzahl Datenbits	4800		9600		38'400		230'400																				
Baudrate	Anzahl Datenbits																															
4800																																
9600																																
38'400																																
230'400																																
7.	Ergänze in folgender Tabelle die Baudrate resp. die Taktdauer.																															
	<table><tr><th>Baudrate</th><th>Taktdauer</th><th>Baudrate</th><th>Taktdauer</th><th>Baudrate</th><th>Taktdauer</th></tr><tr><td></td><td>20ms</td><td>4800</td><td></td><td></td><td>17us</td></tr><tr><td></td><td>3.3ms</td><td>9600</td><td></td><td></td><td>8.68us</td></tr><tr><td></td><td>833us</td><td>19'200</td><td></td><td>230'400</td><td></td></tr><tr><td></td><td>417us</td><td>38'400</td><td></td><td>460'800</td><td></td></tr></table>	Baudrate	Taktdauer	Baudrate	Taktdauer	Baudrate	Taktdauer		20ms	4800			17us		3.3ms	9600			8.68us		833us	19'200		230'400			417us	38'400		460'800		
Baudrate	Taktdauer	Baudrate	Taktdauer	Baudrate	Taktdauer																											
	20ms	4800			17us																											
	3.3ms	9600			8.68us																											
	833us	19'200		230'400																												
	417us	38'400		460'800																												
8.	Erkläre die Start / Stop-Methode.																															
9.	Erkläre warum Fehler bei der Bitübertragung passieren wenn Sender und Empfänger nicht die gleichen Taktfrequenzen (Baudraten) haben). Ein Bit wird immer in der Mitte der Bit-Zeit überprüft. (Skizze machen).																															


	Frage	Lösung	
10.	Erkläre den Begriff synchron und asynchron bei der seriellen Datenübertragung.		
11.	Zähle mindestens 4 Anwendungen für die serielle Datenübertragung auf!		
12.	<p>Studiere im Referenz Manual das Kapitel USART.</p> <p>a) Welches Kapitel beschreibt die UART's?</p> <p>b) Was für Besonderheiten haben die USART's?</p> <p>c) Welche Betriebsarten sind möglich?</p> <p>d) Wieviel Uart's hat der STM32F107VC?</p> <p>e) Welche Bits können im Statusregister abgefragt werden. ?</p> <p>f) Welche Befehle für die serielle Schnittstelle bietet die Touch-Library welche mit dem MCB32 zur Verfügung steht. ?</p> <p>g) Ist es möglich die serielle Schnittstelle USART2 via Interrupt zu steuern?</p> <p>h) Was ist der Unterschied UART- USART?</p>		

A1 Ascii Tabelle

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	 	Space	64	40	100	@	@
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C
4	4	004	EOT (end of transmission)	36	24	044	$	\$	68	44	104	D	D
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G
8	8	010	BS (backspace)	40	28	050	((72	48	110	H	H
9	9	011	TAB (horizontal tab)	41	29	051))	73	49	111	I	I
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[[
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135]	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_

Source: www.LookupTables.com

A2 Abkürzungen

Abkürzung	Begriff	Bedeutung
IDR	Port Input Data Register	
BSRR	Port Bit Set/Reset Register	
BRR	Port Bit Reset Register	
ODR	Port Output Data Register	
CRL	Configuration Register Low	
CRH	Configuration Register High	
GPIO	General purpose input output	
RTEM	Run-Time Environment Manage	Liste der Software Komponenten welche für das Projekt und die MCU zur Verfügung stehen. 
CMSIS	Cortex Microcontroller Software Interface Standard	Der „ <i>Cortex Mikrokontroller Software Interface Standard</i> “ (CMSIS) bietet eine Standard-Plattform für alle Cortex-M Anbieter. Code-Wiederverwendung wird damit unterstützt. Siehe auch: http://www.keil.com/cmsis
AHB	Advanced High Performance Bus	AHB (Advanced High Performance Bus) to APB (Advanced Peripheral Bus) bridge: This bridge divides AHP bus into two buses, APB1 and APB2. APB1 is for peripheral which their frequency is 36 MHz and APB2 is for peripherals which they operate with 72 MHz frequency
APB	Advanced Peripheral Bus	

A3 Anhang Referenzen

- [1] O. Saal, «rs232,» [Online]. Available: <http://www.oliver-saal.de/elektronik/rs232.php>. [Zugriff am 12 02 2016].
- [2] EKomp, «www.elektronik-kompodium.de,» [Online]. Available: <http://www.elektronik-kompodium.de/sites/com/0310301.htm>. [Zugriff am 12 2 2016].
- [3] Malacarne, «MCB32_Befehlsliste_LIB_Vxxyy.pdf,» rma, 2016.
- [4] «RS232-Stopbit,» [Online]. Available: <https://de.wikipedia.org/wiki/Stopppbit>. [Zugriff am 12 2 2016].
- [5] «RS232,» Wiki, [Online]. Available: <https://de.wikipedia.org/wiki/RS-232>. [Zugriff am 2016 2 12].

A4 Programmierung der seriellen Schnittstelle USART2 mit der Touch-Library

Die Programmierung der Schnittstelle kann auch ohne Registerprogrammierung und ohne CMSIS via die Library TouchP0P1 erfolgen. Die entsprechenden Befehle sind auf der nächsten Seite erläutert.

```
/** @brief Serielles Einlesen eines Buchstabens an USART2 PD6
 *          und um 1 erhöht Ausgabe an PD5 via Termini1 an PC
 *          USART Init via spezielle Funktionen der TouchP0P1-Library
 */
//=====Includes=====
#include <stm32f10x.h>           // Mikrocontrollertyp
#include "TouchP0P1.h"          // P0/P1,8Bit,Touchscreen und Grafik

//=====main()=====
int main(void)                  // Hauptprogramm
{
    char c = '0';               // Starte mit Zeichen 0
    USARTInit(2, 0);             // Default: USART2 ohne Intr. mit 9600Bd, 1,8,1,n
    InitTouchP0P1 ("0");

    while(1)                    // Endlosschleife
    {
        if(USARTtoRead(2))      // 1=Zeichen empfangen, 0=kein Zeichen empfangen
        {
            c = USARTRead(2);    // Zeichen lesen von RX
            c = c+1;              // Zeichen +1 an TX
            USARTWrite(2,c);     // Sende Zeichen c
        }
    }
}
```

Der Befehl USARTInit(2,0) startet den USART 2 mit 9600 Baud mit einem Start und einem Stopbit. Somit ist die Schnittstelle bereit. Mit USARTtoRead(2) wird geprüft ob an der Schnittstelle ein Zeichen eingetroffen ist. Siehe Code oben.

A4.1 Auszug aus TouchPOP1 Library Befehlsliste

Die 4 Befehle erlauben ein einfaches Handling der Schnittstelle. Für eine erweiterte Programmierung siehe Kapitel 4.

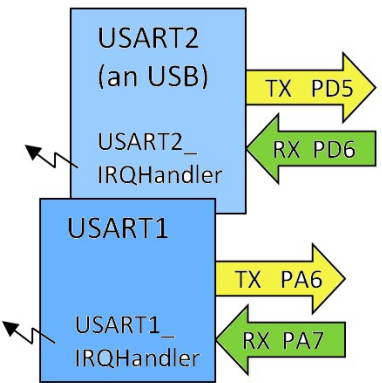
USART1 & USART2 Programmierung

2x USART USART1 TX-> PB6, RX <- PB7 (via Remapping)
 USART2 TX-> PD5, RX <- PD6

Universal Synchronous Asynchronous Receiver Transmitter USART2 ist geeignet für Kommunikation mit PC;

default USART1: 19200Bd, 1,8,1,n //PCLK2 mit 72MHz max (MCB32 nach Reset)

default USART2: 9600Bd, 1,8,1,n //PCLK1 mit 32MHz max (MCB32 nach Reset)

Funktion	Beschreibung / Beispiel	Hinweis zur Hardware
USARTInit (1/2, " IRQPrio0..15);	Nr: 1, 2 Einer von 2 USART IRQPrio: 0, 1..15 Interrupt Priorität; 0 = AUS, 1 höchste, 15 tiefste Aktiviert den UART-Clock und Initialisiert auf 19200Bd resp. 9600, 1Start-, 8 Daten-, 1 Stopbit, No Parity . Bei IRQPrio >0 löst ein ankommendes Zeichen einen Interrupt aus. USARTInit(2, 0); // USART2 ohne Intr.	
USARTWrite (1/2, 'char');	Schreibt den Wert resp. das Zeichen 'CHAR' via USART1 oder 2 auf den seriellen Bus. USARTWrite(2, 'c'); // Sende Zeichen 'c'	
char USARTRead (1 / 2);	Wartet bis ein Zeichen an USART 1 oder 2 eintrifft und gibt Wert via Var. char zurück. c = USARTRead(2); // Warte auf Zeichen ..	
char USARTtoRead (1/2);	Gibt den Status des USART Kanales zurück. 1 = Zeichen eingetroffen. 0= kein Zeichen im Buffer. c = USARTtoRead(2); // 0/1 in c	

Beispiel: neue Baudrate für USART 1 programmieren (mehr im Kapitel 4.)

```

USARTInit(1,0);           // Schalte USART1 mit 19200Bd ein (Default), ohne IR
USARTInit(2,0);           // Schalte USART1 mit 9600Bd ein (Default), ohne IR
USART1->BRR = 0x1D4C;     // USART1: 9600Bd @ 72MHz benötigt einen Teiler von 468.75
                           // siehe RefManual Page 792

```

A5 Auftrag: Serielle Schnittstelle

Name / Datum:	Datum: _____		Punkte
Absicht	Ziel und Weg	Serielle Schnittstelle verstanden und mit Hilfe einer Zusammenfassung (Spick) das Wissen weiterzugeben (Vortrag). Sie sind in der Lage ein einfaches Kommunikationsprogramm zu entwerfen und dies zu demonstrieren.	
Auftrag			
	Studium	Lese den Auftrag (Kap.: 6) und Plane das Vorgehen: Studiere das aktuelle Skript mit den schon gelösten Aufgaben. Fasse das Gelesene kurz , präzis und einfach formuliert zusammen.	Plan 3 ZF 6
	Entwerfen	Plane das Erstellen der SW. Struktogramm Skizzen. Mind-Map.	
	Entwickeln Ausarbeiten	Wie kann der PC mit dem MCB kommunizieren? Hyperterminal? Erstelle das Programm nach den gelernten Regeln. Dokumentation vollständig im Programmcode (Header, Code).	6 6
	Zusammenfassen	Überprüfe und vervollständige die Zusammenfassung (Q1).	
	Präsentieren	Präsentiere die Lösung dem Lehrer. Speichere die Lösung im Namensdirectory unter 20160225 ab	6 2
Optionen	Opt 1:	2 MCB32 miteinander verbinden. Wie geht das?	+2
	Opt 2:	Verbessere die Programme so, dass sie mit Interrupts arbeiten.	+2
	Opt 3:	Anstelle eines Terminal-Programmes soll ein eigenes Programm entwickelt werden. Vorlagen vom Lehrer.	+2
	Opt 4:	Sende vom PC ein Datenfile mit X-Y Daten (CSV oder TXT Format) und Stelle die Daten auf dem MCB32 grafisch dar.	+4
Auswertung	Excellent (6)	Punkte: (max 29 ohne +Punkte und ohne Q's)	
	Very Good (5)	(Q1) Qualität Zusammenfassung:	6
	Good (4)	(Q2) Qualität Präsentation:	6
	Average (3)	(Q3) Qualität Code/Dokumentation:	6
	Poor (2)		
	Kommentar:		
Punkte:		Note:	





A7 Lösungen Serial 1 ... Serial 4

