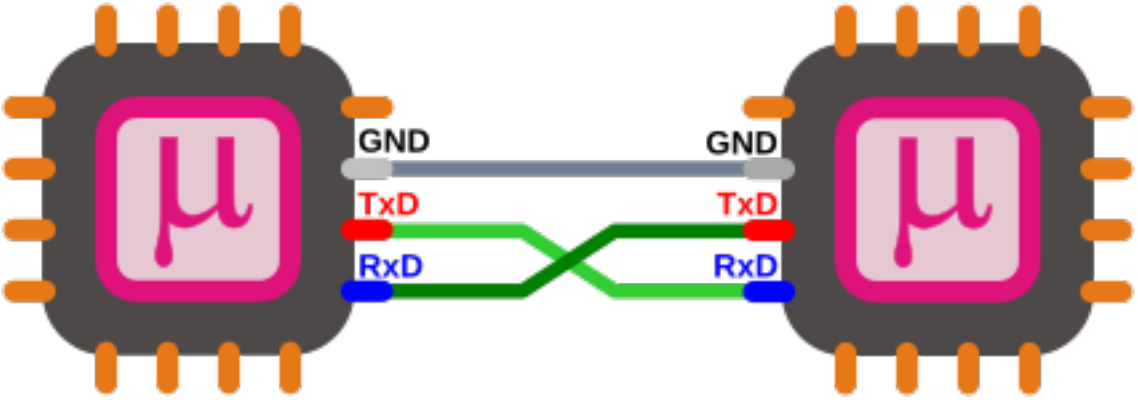
DIC-Serielle Kommunikation mit einem µC

Fabio Plunser

1. Dezember 2020



Inhaltsverzeichnis

1 Aufgabenstellung **1**

2 RS485 **2**
2.1 Generelles 2
2.2 Wie funktioniert es? 2
2.3 Wie wird der MAX485 angeschlossen 3

3 Einteilung — Was man wissen muss **5**

4 Clock/Baudrate der UART **6**
4.1 Baudrate-Berechnung-Register-Setzen 7
4.2 Weitere USART einstellungen 8
4.3 USART-Initiallisierung/Konfiguration—Code 8

5 GPIO **9**
5.1 GPIO-Erklärung 9
5.2 GPIO-Code 10

6 Gesamtes-Programm **11**

7 Anhang **13**
7.1 Verlinkungen 13

Abbildungsverzeichnis

1 RS485-Diagramm 2
2 MAX485-Arduino-Anschluss-BSP 3
3 STM32F030F4P6-Pinout 4
4 Clock-Tree 6
5 USART-Aufbaug 7
6 System-Architektur 9
7 GPIO-UART-PIN-Table 9
8 GPIO-AF 10
9 GPIO-AF-Register 10

Code

1 Init-UART 8
2 GPIO-Code 10
3 Gesamter-Code 11

1 Aufgabenstellung

Die Aufgabe ist es auf dem STM32F030F4 Chip die UART3 so zu programmieren, dass sie den Wert eines eingebauten ADC per interrupt ausgibt.
Die richtige Aufgabe ist es die STM32F030F4 UART3 zu programmieren dass sie wenn sie ein zeichen erhält 10 Bytes zurückschickt.
Da dieses spezielle Package des STM32 keine UART3 besitzt ist die Aufgabenstellung so nicht möglich somit wird einfach die vorhandene UART1 Schnittstelle verwendet.
Der STM UART Ausgang wird mit einem MAX485 auf RS485 übersetzt.
UART einstellungen:
Baudrate: 38400 mit ODD parity

2 RS485

2.1 Generelles

Ist ein Industriestandard der eine asynchrone serielle Datenübertragung ermöglicht.
Der Standard verwendet ein symmetrisches Leitungspaar, dass für eine höhere elektromagnetische Resistenz sorgt.

2.2 Wie funktioniert es?

Betriebsspannung 5V oder 3.3V
Der empfänger wertet die die Differenz beider Leitungen aus und kann Pegel ab $\pm 200mV$ erkennen.
Senderpegel können von $\pm 1.5V bis \pm 6V$
Logik:
Wenn $U_+ - U_- < -0.3V$ = MARK = OFF = Logisch 1
Wenn $U_+ - U_- > +0.3V$ = SPACE = ON = Logisch 0

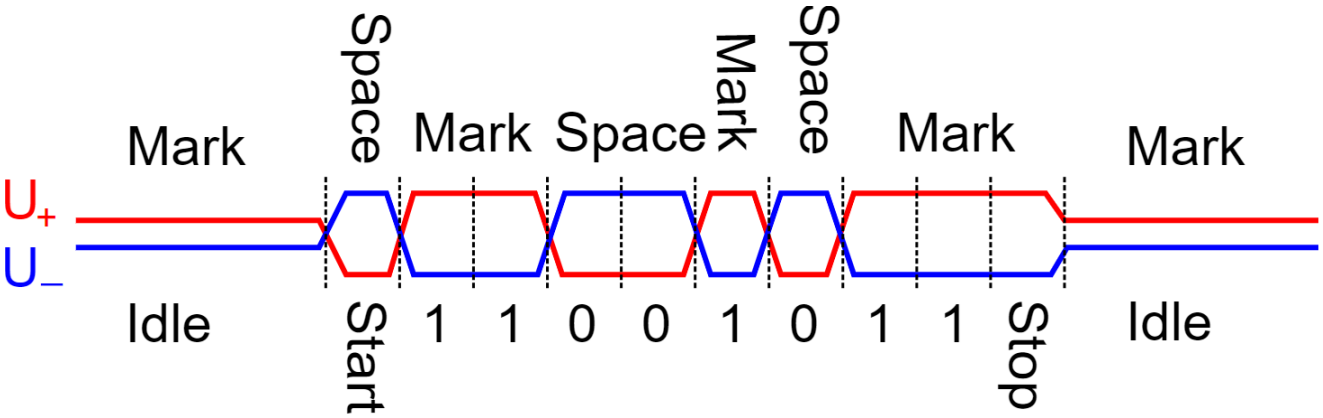


Abbildung 1: RS485-Diagramm

2.3 Wie wird der MAX485 angeschlossen

Unten ist ein Beispiel mit einem Arduino UNO
Anschluss:
DI → TX
RO → RX
DE, RE auf einen GPIO Pin. Da wenn DE + RE = 1 → Daten können nur gesendet werden.
Wenn DE + RE = 0 → Daten können nur empfangen werden.

Ebenso muss der

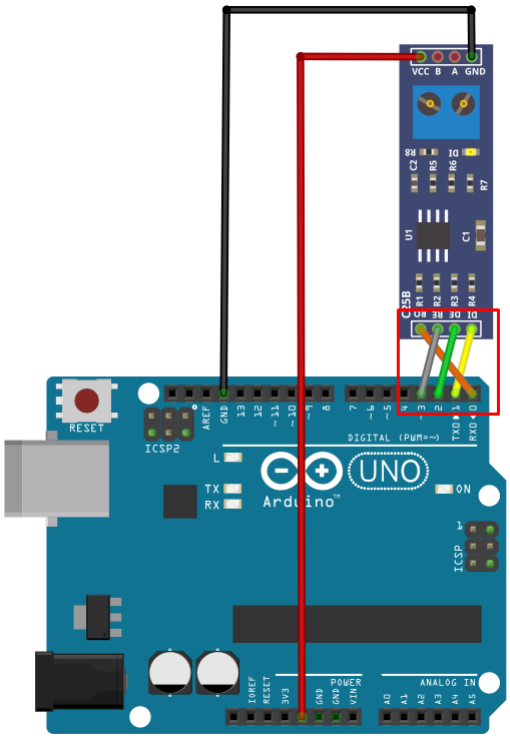


Abbildung 2: MAX485-Arduino-Anschluss-BSP

Ebenso muss dann der MAX485 and dem STM angeschlossen werden. Das untere Bild ist das Demo Board des STM32F030F4, eines der wenigen Boards, dass man zum Testen des Programmes für den STM Chip kaufen kann.

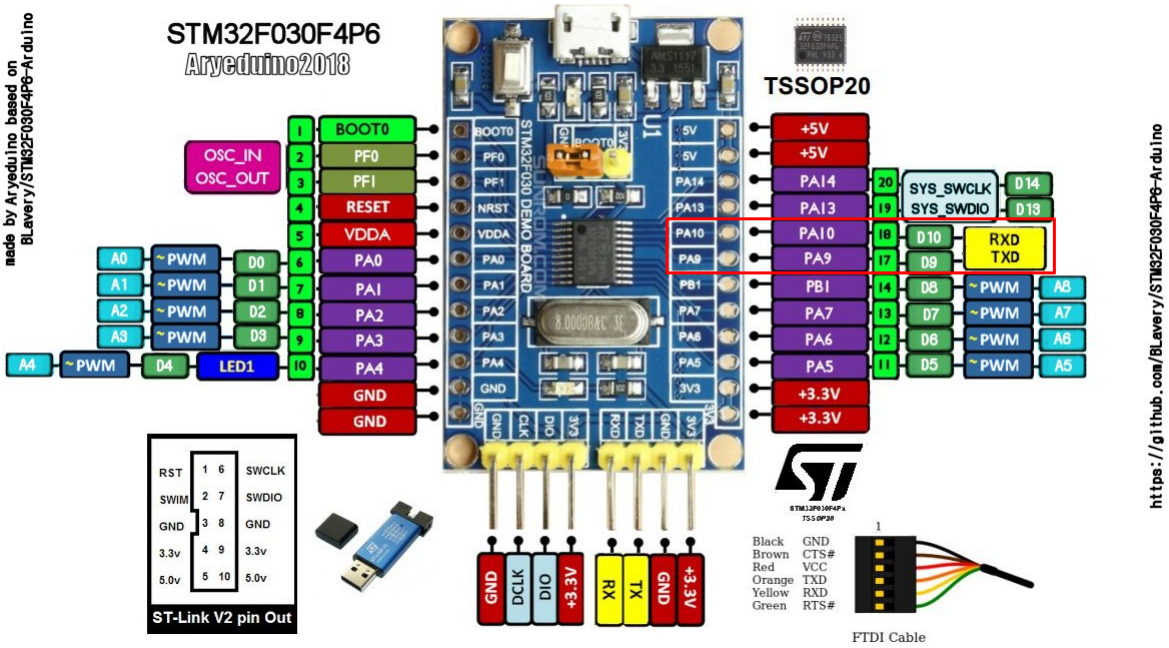


Abbildung 3: STM32F030F4P6-Pinout

3 Einteilung — Was man wissen muss

- Clock aufbau / einstellen
- GPIO Pins register finden / GPIO Pins einstellen
- UART Register suchen / UART aktivieren und einstellen
- Verstehen wie NVIC funktioniert

4 Clock/Baudrate der UART

Die Clock muss passend aus der gewollten Baudrate für die UART ausgewählt werden damit die Baudrate richtig berechnet wird.
Standard Clock = 8MHz
Die Uart liegt im Adressraum **0x4001 3800 - 0x4001 3BFF**

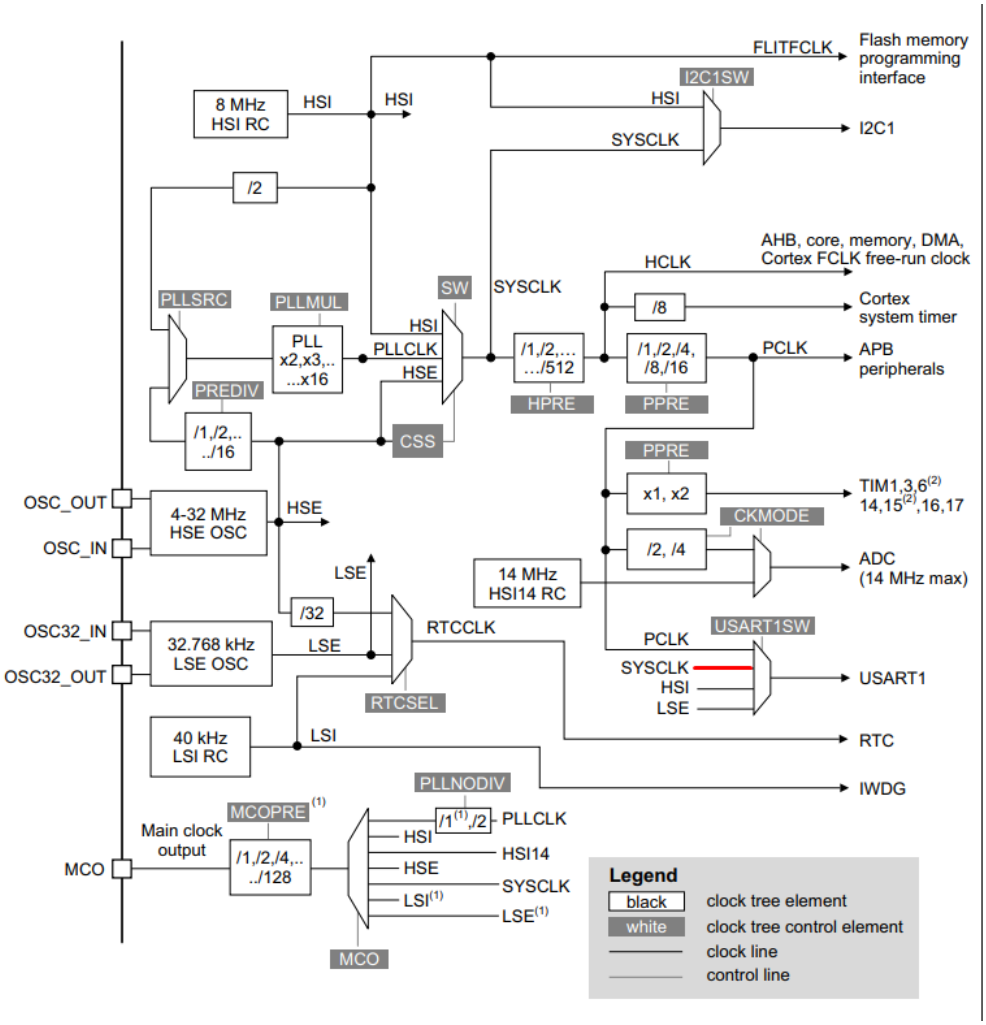


Abbildung 4: Clock-Tree

4.1 Baudrate-Berechnung-Register-Setzen

Um die Baudrate einstellen zu können muss in das Register **USART_BRR** die richtige Hexadezimal Zahl geschrieben werden.

Berechnung: $\frac{Clock}{Baudrate}$, $\frac{8MHZ}{Baudrate} = 200_{dezimal}$ bzw. $D0_{hex}$

Da der Systemclock gleich der HSI clock ist, kann dafür im Register **RCC_CFGR3 bei der Addresse: 0x30** die USART1SW entweder mit 01 für Systemclock oder mit 11 für HSI clock beschrieben werden. Zusätzlich muss im Register **USART_BRR mit Address offset: 0x0C** D0 geschrieben Werden.

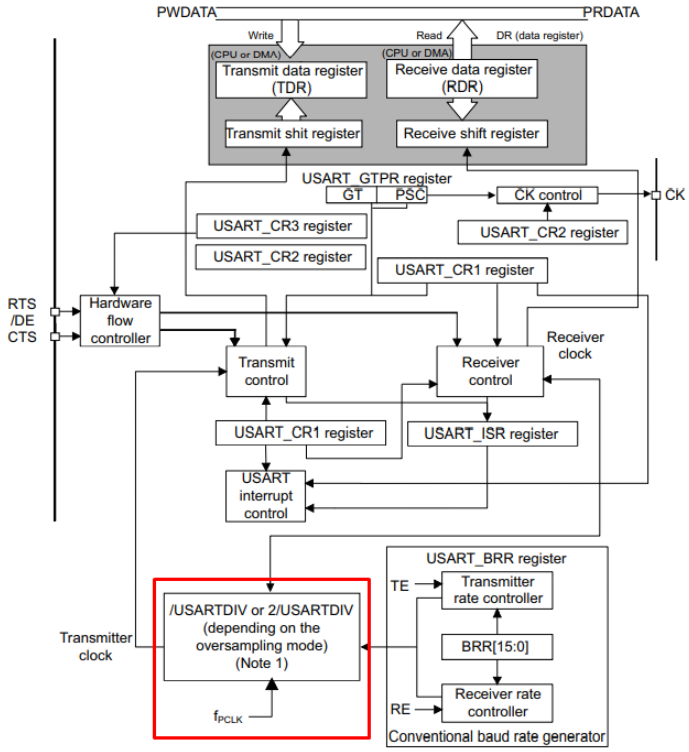


Abbildung 5: USART-Aufbaug

4.2 Weitere USART einstellungen

Die Eistellungen der USART müssen getroffen werden, bevor sie aktiviert wird. Laut angabe wird noch eine ODD Parity verwendet diese kann in dem Register **USART_CR1** auf Bit9: festgelegt werden. Weiterhin müssen dort unter Bit3 und Bit2, TX und RX aktivieren.

4.3 USART-Initiallisierung/Konfiguration—Code

```
int init_UART()
{
    u_int32_t* USART_CR1 = (u_int32_t *)0x40013800+0x00;;
    u_int32_t* RCC_CFGR3 = (u_int32_t *)0x30;
    u_int32_t* USART_BRR = (u_int32_t *)0x4001+0x0C;
    u_int32_t REG_CONTENT;

    //Use SYSCLK for USART and use Baudrate 38400
    REG_CONTENT = *RCC_CFGR3;
    REG_CONTENT |= 0x0000002;
    *RCC_CFGR3 = REG_CONTENT;

    REG_CONTENT = *USART_BRR;
    REG_CONTENT = 0x0000D0;
    *USART_BRR = REG_CONTENT;

    //Wordlenght, Parity control enable, Parity selection,
       interrupt enable, Transmission complete interrupt enable,
        RXNE interrupt enable
    REG_CONTENT = *USART_CR1;
    REG_CONTENT |= 0x000006EC;
    *USART_CR1 = REG_CONTENT;

    REG_CONTENT = *USART_CR1;
    REG_CONTENT |= 0x00000001; //enable UART
    *USART_CR1 = REG_CONTENT;
    Test
}
```

Listing 1: Init-UART

5 GPIO

5.1 GPIO-Erklärung

Nun da die UART richtig eingestellt ist muss für die Kommunikation die GPIO Pins konfiguriert werden.
Wie man bei der Abbildung 6 sehen kann, sind die GPIOs am BUS AHB2 verbunden. Da das verwendete Paket nur GPIO A verwendet müssen dementsprechend die GPIO-A Register richtig konfiguriert werden
Das Register um die GPIOs als output einzustellen ist **GPIOA_MODER im Address-raum 0x4800 0000 (des AHB2 Buses) mit dem Address offset: 0x00**
Um Pins für die UART verwenden zu können müssen diese Pins noch als älternate functi-ons"konfiguriert werden im Register **GPIOA_AFRH mit Address offset: 0x24** Welche Pin Nummer man Programmieren muss sieht man in der Abbildung 7, man benötigt PA9 (Pin:17) und PA19 (Pin:18), da diese als alternate function die USART_1 TX und RX hinterlegt haben. Wie das alternate function Register konfiguriert werden muss sieht ba in den Abbildungen 8 und 9

Weiterhin müssen noch zwei GPIO Pins Für die DE und RE Pins des MAX485 als output konfiguriert werden. DA PA7 und PA8 am nähesten dran sind an den anderen Pins verwende ich diese.

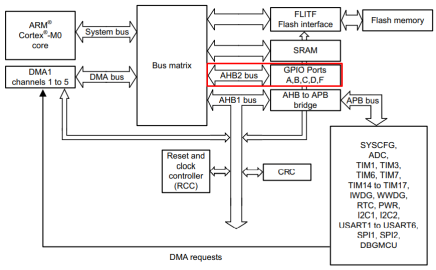


Abbildung 6: System-Architektur

Pin number				Pin name (function after reset)	Pin type	I/O structure	Notes	Pin functions	
LQFP64	LQFP48	LQFP32	TSSOP20					Alternate functions	Additional functions
34	26	-	-	PB13	I/O	FT	-	SPI1_SCK(2), SPI2_SCK(3)(5), I2C2_SCL(5), TIM1_CH1N, USART3_CTS(5)	-
35	27	-	-	PB14	I/O	FT	-	SPI1_MISO(2), SPI2_MISO(3)(5), I2C2_SDA(5), TIM1_CH2N, TIM15_CH1(3)(5), USART3_RTS(5)	-
36	28	-	-	PB15	I/O	FT	-	SPI1_MOSI(2), SPI2_MOSI(3)(5), TIM1_CH3N, TIM15_CH1N(3)(5), TIM15_CH2(3)(5)	RTC_REFIN, WKUP7(5)
37	-	-	-	PC6	I/O	FT	-	TIM3_CH1	-
38	-	-	-	PC7	I/O	FT	-	TIM3_CH2	-
39	-	-	-	PC8	I/O	FT	-	TIM3_CH3	-
40	-	-	-	PC9	I/O	FT	-	TIM3_CH4	-
41	29	18	-	PA8	I/O	FT	-	USART1_CK, TIM1_CH1, EVENTOUT, MCO	-
42	30	19	17	PA9	I/O	FT	-	USART1_TX, TIM1_CH2, TIM15_BKIN(3)(5) I2C1_SCL(2)(5)	-
43	31	20	18	PA10	I/O	FT	-	USART1_RX, TIM1_CH3, TIM17_BKIN I2C1_SDA(2)(5)	-
44	32	21	-	PA11	I/O	FT	-	USART1_CTS, TIM1_CH4, EVENTOUT, I2C2_SCL(5)	-
45	33	22	-	PA12	I/O	FT	-	USART1_RTS, TIM1_ETR, EVENTOUT, I2C2_SDA(5)	-

Abbildung 7: GPIO-UART-PIN-Table

Table 12. Alternate functions selected through GPIOA_AFR registers for port A

Pin name	AF0	AF1	AF2	AF3	AF4	AF5	AF6
PA0	-	USART1_CTS(2)	-	-	USART4_TX(1)	-	-
		USART2_CTS(1)(3)					
PA1	EVENTOUT	USART1_RTS(2)	-	-	USART4_RX(1)	TIM15_CH1N(1)	-
		USART2_RTS(1)(3)					
PA2	TIM15_CH1(1)(3)	USART1_TX(2)	-	-	-	-	-
		USART2_TX(1)(3)					
PA3	TIM15_CH2(1)(3)	USART1_RX(2)	-	-	-	-	-
		USART2_RX(1)(3)					
PA4	SPI1_NSS	USART1_CK(2)	-	-	TIM14_CH1	USART6_TX(1)	-
		USART2_CK(1)(3)					
PA5	SPI1_SCK	-	-	-	-	USART6_RX(1)	-
PA6	SPI1_MISO	TIM3_CH1	TIM1_BKIN	-	USART3_CTS(1)	TIM16_CH1	EVENTOUT
PA7	SPI1_MOSI	TIM3_CH2	TIM1_CH1N	-	TIM14_CH1	TIM17_CH1	EVENTOUT
PA8	MCO	USART1_CK	TIM1_CH1	EVENTOUT	-	-	-
PA9	TIM15_BKIN(1)(3)	USART1_TX	TIM1_CH2	-	I2C1_SCL(1)(2)	MCO(1)	-
PA10	TIM17_BKIN	USART1_RX	TIM1_CH3	-	I2C1_SDA(1)(2)	-	-
PA11	EVENTOUT	USART1_CTS	TIM1_CH4	-	-	SCL	-

Abbildung 8: GPIO-AF

8.4.10 **GPIO alternate function high register (GPIOx_AFRH) (x = A..D, F)**

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFSEL15[3:0]				AFSEL14[3:0]				AFSEL13[3:0]				AFSEL12[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFSEL11[3:0]				AFSEL10[3:0]				AFSEL9[3:0]				AFSEL8[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **AFSELy[3:0]:** Alternate function selection for port x pin y (y = 8..15)
These bits are written by software to configure alternate function I/Os

AFSELy selection:

0000: AF0
0001: AF1
0010: AF2
0011: AF3
0100: AF4
0101: AF5
0110: AF6
0111: AF7
1000: Reserved
1001: Reserved
1010: Reserved
1011: Reserved
1100: Reserved
1101: Reserved
1110: Reserved
1111: Reserved

Abbildung 9: GPIO-AF-Register

5.2 GPIO-Code

```
int GPIO()
{
    //Set GPIO pins PA9 and PA10 alternate function for USART TX and
        RX, set PA7 and PA6 output for DE and RE of MAX485
    u_int32_t* GPIOA_MODER = (u_int32_t*)0x400000+0x00;
    u_int32_t* GPIOA_AFRH  = (u_int32_t*)0x400000+0x24;
    u_int32_t REG_CONTENT;

    REG_CONTENT = *GPIOA_MODER;
    REG_CONTENT |= 0x285000;
    *GPIOA_MODER = REG_CONTENT;

    //Set GPIO PA9 as AF=TX and PA10 as AF = RX
    REG_CONTENT = *GPIOA_AFRH;
    REG_CONTENT |= 0x110;
    *GPIOA_AFRH = REG_CONTENT;
}
```

Listing 2: GPIO-Code

6 Gesamtes-Programm

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include <unistd.h>
#include <fcntl.h>
#include <bits/types.h>

#define USART1_interrupt (u_int32_t*)0x000000AC

int init_GPIO()
{
    //Set GPIO pins PA9 and PA10 alternate function for USART TX and RX, set
        PA7 and PA6 output for DE and RE of MAX485
    u_int32_t* GPIOA_MODER = (u_int32_t*)0x400000+0x00;
    u_int32_t* GPIOA_AFRH  = (u_int32_t*)0x400000+0x24;
    u_int32_t REG_CONTENT;

    REG_CONTENT = *GPIOA_MODER;
    REG_CONTENT |= 0x285000;
    *GPIOA_MODER = REG_CONTENT;

    //Set GPIO PA9 as AF=TX and PA10 as AF = RX
    REG_CONTENT = *GPIOA_AFRH;
    REG_CONTENT |= 0x110;
    *GPIOA_AFRH = REG_CONTENT;


}

int init_UART()
{
    u_int32_t* USART_CR1 = (u_int32_t *)0x40013800+0x00;
    u_int32_t* RCC_CFGR3 = (u_int32_t *)0x30;
    u_int32_t* USART_BRR = (u_int32_t *)0x4001+0x0C;
    u_int32_t REG_CONTENT;

    //Use SYSCLK for USART and use Baudrate 38400
    REG_CONTENT = *RCC_CFGR3;

    REG_CONTENT |= 0x00000001;
    *RCC_CFGR3 = REG_CONTENT;

    *UUSART_BRR = REG_CONTENT
    REG_CONTENT = *USART_BRR;
    REG_CONTENT = 0x0000D0;
    *USART_BRR = REG_CONTENT;

    //Wordlenght, Parity control enable, Parity selection, interrupt enable,
        Transmission complete interrupt enable, RXNE interrupt enable
```

```
    REG_CONTENT = *USART_CR1;
    REG_CONTENT |= 0x000006EC;
    *USART_CR1 = REG_CONTENT;

    REG_CONTENT = *USART_CR1;
    REG_CONTENT |= 0x00000001; //enable UART
    *USART_CR1 = REG_CONTENT;
}

int main()
{
    init_UART();
    init_GPIO();

    for(;;)
    {

    }
}
```

Listing 3: Gesamter-Code

7 Anhang

7.1 Verlinkungen

Abbildung: ??:
`http://www.mathe-mit-methode.com/schlaufuchs_web/elektrotechnik/mikrocontroller_lernmaterial/microcontroller_allgemein/mikrocontroller_ext_hardware/mikrocontroller_uart_bild_001.html`

Abbildung: 1
`https://de.wikipedia.org/wiki/EIA-485`

Abbildung: 3
https://user-images.githubusercontent.com/20950920/48240567-e985c080-e3db-11e8-8775-68a216485b59.jpg von aryeguetta