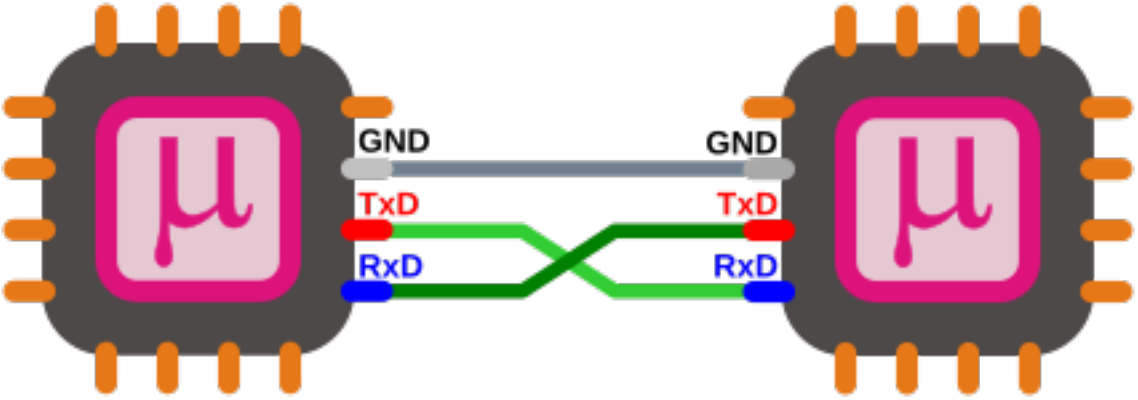
DIC-Serielle Kommunikation mit einem µC

Fabio Plunser

31. 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 GPIO **6**
4.1 GPIO-Erklärung . . . 6
4.2 GPIO-Code . . . 7

5 Clock/Baudrate der UART **8**
5.1 Baudrate-Berechnung-Register-Setzen . . . 9
5.2 Weitere USART einstellungen . . . 10
5.3 USART-Initiallisierung/Konfiguration—Code . . . 11

6 ADC **12**
6.1 Erklärung . . . 12
6.2 Register . . . 12
6.3 Konfiguration . . . 12
6.4 ADC-Initiallisierungs-Code . . . 13
6.5 ADC-Interrupt-Handler-Code . . . 14

7 NVIC **15**
7.1 Wir wird er verwendet . . . 15
7.2 NVIC-Code . . . 15

8 Gesamtes-Programm **16**
8.1 Headerfile . . . 16
8.2 Main . . . 20

9 Anhang **25**
9.1 Verlinkungen . . . 25

Abbildungsverzeichnis

1 RS485-Diagramm . . . 2
2 MAX485-Arduino-Anschluss-BSP . . . 3
3 STM32F030F4P6-Pinout . . . 4
4 System-Architektur . . . 6
5 GPIO-UART-PIN-Table . . . 6
6 GPIO-AF . . . 7
7 GPIO-AF-Register . . . 7
8 Clock-Tree . . . 8
9 USART-Aufbaug . . . 9
10 USART_BRR . . . 10
11 NVIC-Vector-Table . . . 15
12 NVIC-ISER-Register . . . 15

Code

1 GPIO-Code . . . 7
2 Init-UART . . . 11
3 ADC-Initiallisierungs-Code . . . 13
4 ADC-Interrupt-Handler-Code . . . 14
5 NVIC-enable-interrupts . . . 15
6 Headerfile . . . 16
7 Gesamter-Code . . . 20

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

Dieser Arbeitsauftrag kann in dieser GIT-Repo verfolgt werden: `https://github.com/FabioPlunser/DIC-Lezuo`

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

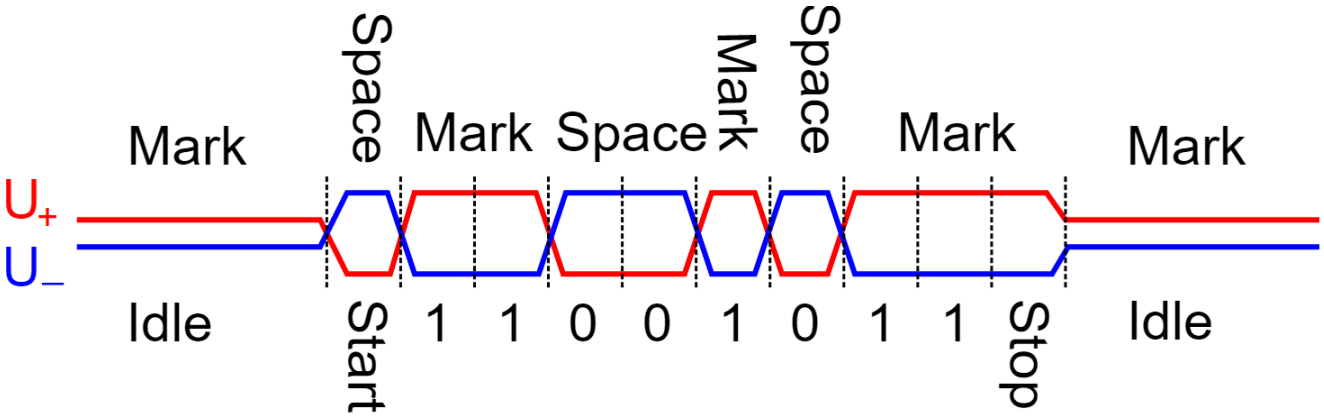


Abbildung 1: RS485-Diagramm

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.

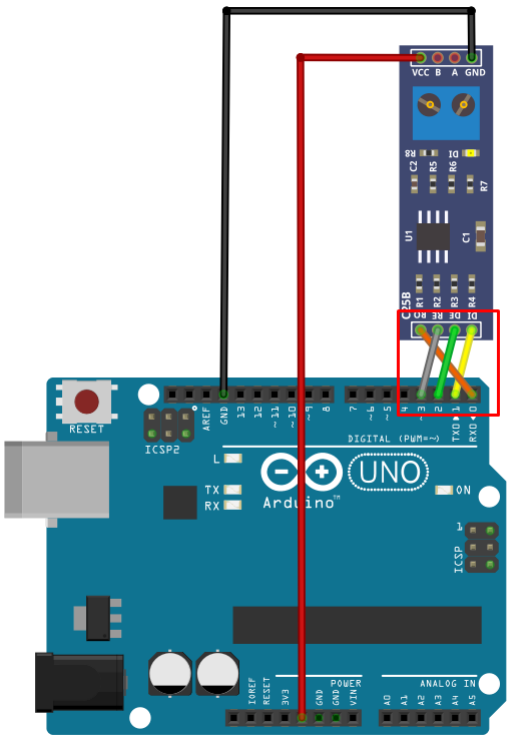


Abbildung 2: MAX485-Arduino-Anschluss-BSP

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.

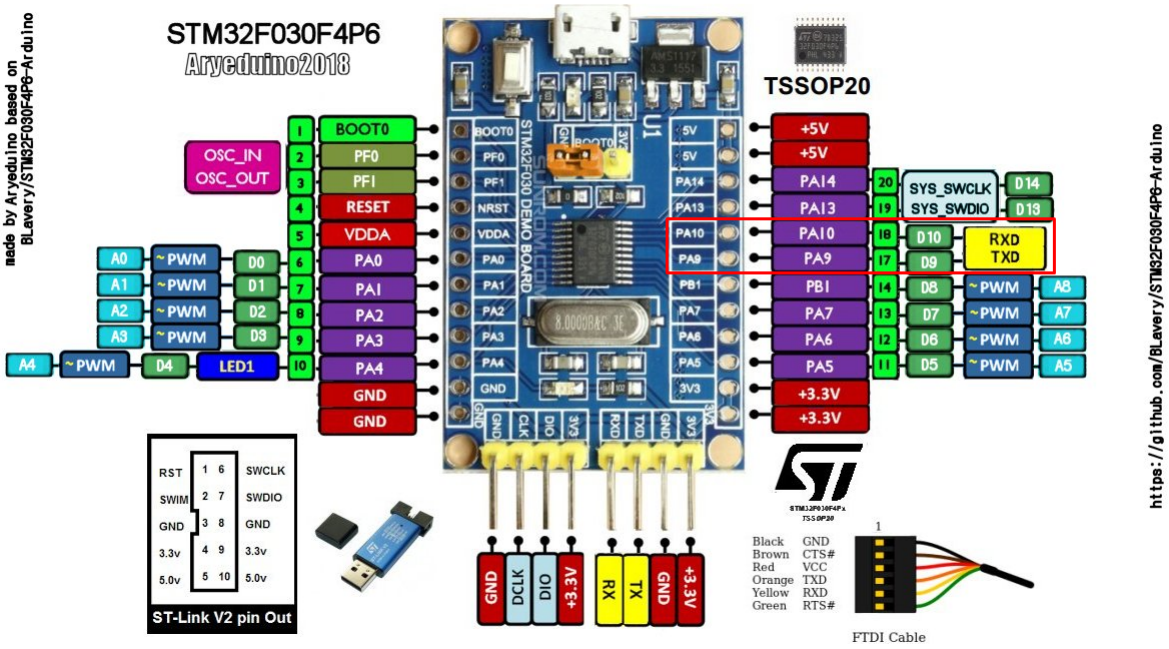


Abbildung 3: STM32F030F4P6-Pinout

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 GPIO

4.1 GPIO-Erklärung

Nun da die UART richtig eingestellt ist muss für die Kommunikation die GPIO Pins konfiguriert werden.
Wie man bei der Abbildung 4 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 Addressraum 0x4800 0000 (des AHB2 Buses) mit dem Address offset: 0x00**
Um Pins für die UART verwenden zu können müssen diese Pins noch als älternate functions"konfiguriert werden im Register **GPIOA_AFRH mit Address offset: 0x24** Welche Pin Nummer man Programmieren muss sieht man in der Abbildung 5, 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 6 und 7

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. Ebenfalls wird ein GPIO Pin als analog konfiguriert um diesen als ADC Eingang zu verwenden.

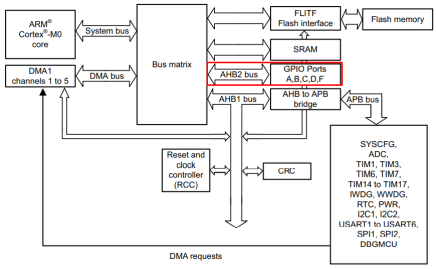


Abbildung 4: System-Architektur

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 5: 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 6: 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 7: GPIO-AF-Register

4.2 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
    //PA0 ADC_IN0 so set it to analog and then in DAC register
    //set PA0 as ADC_IN0
    uint32_t REG_CONTENT;
    uint32_t* gpioa_moder = GPIOA_MODER;
    uint32_t* gpioa_afrh = GPIOA_AFRH;
    uint32_t* gpioa_odr = GPIOA_ODR;

    REG_CONTENT = *gpioa_moder;
    REG_CONTENT |= 0x00285003;
    *gpioa_moder = REG_CONTENT;

    //Set GPIO PA9 as AF=TX and PA10 as AF = RX
    REG_CONTENT = *gpioa_afrh;
    REG_CONTENT |= 0x00000110;
    *gpioa_afrh = REG_CONTENT;

    //Ensure that output gpios are 0 for MAX, to read all the time
    //and only send, when needing to send
    REG_CONTENT = *gpioa_odr;
    REG_CONTENT |= 0;
    *gpioa_odr = REG_CONTENT;
}
```

Listing 1: GPIO-Code

5 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**

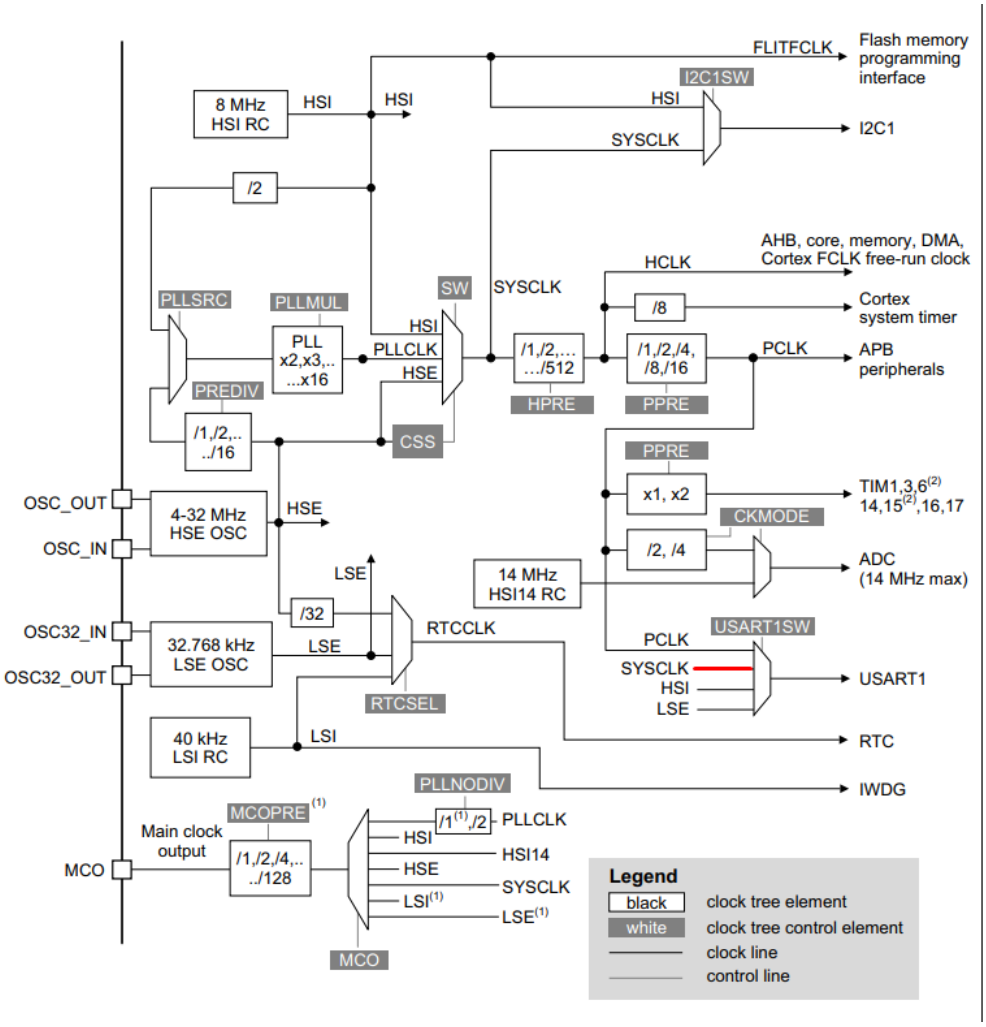


Abbildung 8: Clock-Tree

5.1 Baudrate-Berechnung-Register-Setzen

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

Berechnung: $\frac{Clock}{Baudrate}$, $\frac{8MHz}{38400} = 208{,}3_{dezimal}$ bzw. $D0_{hex}$
Zurückgerechnet $208 * 38400 = 7.98MHz$. $\frac{8MHz}{208}$ =Baudrate von 38461.

Da der Systemclock gleich der HSI clock ist, kann dafür im Register **RCC_CFGR3 bei der Addresse offset: 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.

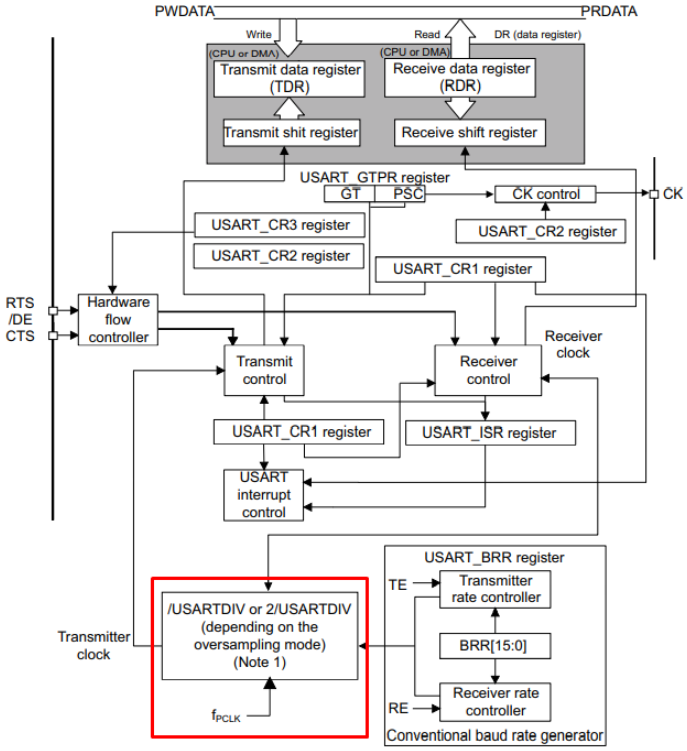


Abbildung 9: USART-Aufbaug

23.7.4 Baud rate register (USART_BRR)

This register can only be written when the USART is disabled (UE=0). It may be automatically updated by hardware in auto baud rate detection mode.

Address offset: 0x0C

Reset value: 0x0000

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

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:4 **BRR[15:4]**
BRR[15:4] = USARTDIV[15:4]

Bits 3:0 **BRR[3:0]**
When OVER8 = 0, BRR[3:0] = USARTDIV[3:0].
When OVER8 = 1:
BRR[2:0] = USARTDIV[3:0] shifted 1 bit to the right.
BRR[3] must be kept cleared.

Abbildung 10: USART_BRR

5.2 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.

5.3 USART-Initiallisierung/Konfiguration—Code

```
int init_UART()
{
    uint32_t REG_CONTENT;
    uint32_t* rcc_cfr3 = RCC_CFGR3;
    uint32_t* usart1_brr = USART1_BRR;
    uint32_t* usart1_cr1 = USART1_CR1;

    //Use SYSCLK for USART and use Baudrate 38400
    REG_CONTENT = *rcc_cfr3;
    REG_CONTENT |= 0x00000001;
    *rcc_cfr3 = REG_CONTENT;

    REG_CONTENT = *usart1_brr;
    REG_CONTENT = 0x00000D00;
    *usart1_brr = REG_CONTENT;

    //Wordlenght, Parity control enable, Parity selection,
    //interrupt enable, Transmission complete interrupt enable,
    //RXNE interrupt enable
    REG_CONTENT = *usart1_cr1;
    REG_CONTENT |= 0x000006EC;
    *usart1_cr1 = REG_CONTENT;

    REG_CONTENT = *usart1_cr1;
    REG_CONTENT |= 0x00000001; //enable UART
    *usart1_cr1 = REG_CONTENT;
}
```

Listing 2: Init-UART

6 ADC

6.1 Erklärung

Der STM hat einen ADC eingebaut, dieser kann bei den größeren Packages sogar als Temperatursensor verwendet werden. Alle GPIOs können als ADC Eingang verwendet werden solange sie als analog GPIO konfiguriert werden.

6.2 Register

Die Register für den ADC beginnen bei 0x4001 2400 bis 0x4001 27FF. Für die Konfiguration werden folgende Register benötigt:

- ADC_ISR — ADC Interrupts, für ADC Ready und ADC End of Convertion flag
- ADC_IER — ADC enable Interrupts
- ADC_CR — ADC kalibrieren, aktivieren, starten
- ADC_CFGR2 — Clock Einstellung
- ADC_CHSELR — ADC Input Channel
- ADC_DR — 16 Bit Ergebniss der Umnwandlung

6.3 Konfiguration

Um den ADC zu konfigurieren, wird zuerst der ADC Clock eingestellt → der ADC kalibriert → ADC aktiviert → GPIO als ADC Input einstellen.
Wenn die Kalibrierung fertig ist, ist das ADCAL Bit 0. Dann kann der ADC und Interrupts aktiviert werden und eingestellt werden welcher GPIO Pin als Input verwendet wird.

Danach wenn der ADC bereit ist wird im Interrupt Register die ADC_ADRDY (ADC Ready) flag gesetzt, dieses wird im ADC-Interrupt-Handler abgefragt und dann eine Umwandlung gestartet.
Wenn eine Umwandlung fertig ist wird im Interrupt Register das ADC_TC (Transmission complete) flag gesetzt. Im ADC-Interrupt-Hanlder wird dann aus dem ADC_DR Register das Ergebniss der Umwandlung weitergeben um dies dann an der USART senden zu können.

6.4 ADC-Initiallisierungs-Code

```
int init_ADC()
{
    uint32_t REG_CONTENT;
    uint32_t* adc_chselr = ADC_CHSELR;
    uint32_t* adc_isr = ADC_ISR;
    uint32_t* adc_cr = ADC_CR;
    uint32_t* adc_cr_adcal = ADC_CR_ADCAL;
    uint32_t* adc_ier = ADC_IER;
    uint32_t* adc_cfgr2 = ADC_CFGR2;
    uint32_t adc_isr_adrdy = ADC_ISR_ADRDY;


    //set ADC clock to PCLK so thats is synchronous with
       sysclock
    REG_CONTENT = *adc_cfgr2;
    REG_CONTENT |= 0x40000000;
    *adc_cfgr2 = REG_CONTENT;

    //before starting callibrate ADC
    REG_CONTENT = *adc_cr;
    REG_CONTENT |= 0x80000000;
    *adc_cr = REG_CONTENT;


    //if calibration is complete enable ADC
    //enable Interrupts
    //set correct channel
    //when ADC is ready, a ad ready interrupt occurs and the
       interrupt handler
    //will then start the ADC convertion
    if((*adc_cr & *adc_cr_adcal) == 0)
    {
        //enable ADC and ensure ADSTART=0 for further
           configuration
        REG_CONTENT = *adc_cr;
        REG_CONTENT |= 0x00000005;
        *adc_cr = REG_CONTENT;

        //enable Interrupts of ADC
        REG_CONTENT = *adc_ier;
        REG_CONTENT |= 0x00000005;
        *adc_ier = REG_CONTENT;

        //Set ADC Channel to channel 0 because PA0 is ADC_IN0
        REG_CONTENT = *adc_chselr;
        REG_CONTENT |= 0x00000001;
        *adc_chselr = REG_CONTENT;
    }
}
```

Listing 3: ADC-Initiallisierungs-Code

6.5 ADC-Interrupt-Handler-Code

```
void ADC1_IRQHandler(void)
{
    uint32_t* adc_isr = ADC_ISR;
    uint32_t* adc_cr = ADC_CR;

    uint32_t* adc_isr_eoc = ADC_ISR_EOC;
    uint32_t* adc_isr_adrdy = ADC_ISR_ADRDY;
    uint32_t* adc_dr = ADC_DR;
    uint32_t* adc_dr_data = ADC_DR_DATA;
    uint32_t* gpioa_odr = GPIOA_ODR;
    uint16_t* usart1_tdr =  USART1_TDR;

    uint32_t adc_cr_adstart = ADC_CR_ADSTART;
    uint32_t ADC_Value;
    uint32_t REG_CONTENT;

    //if ADC Ready start convertion
    if((*adc_isr & *adc_isr_adrdy)== 1)
    {
        REG_CONTENT = *adc_cr;
        REG_CONTENT |= adc_cr_adstart;
        *adc_cr = REG_CONTENT;

    }
    //if convertion complete send ADC value
    if((*adc_isr & *adc_isr_eoc) == 1)
    {
        ADC_Value = (*adc_dr & *adc_dr_data);
        //Set PA7 and PA6 high for max to send data
        REG_CONTENT = *gpioa_odr;
        REG_CONTENT |= 0x000000C0;
        *gpioa_odr = REG_CONTENT;

        USART_write_data = (char*)ADC_Value;
        //write something into the USART Buffer for USART
        //interrupt where rest of adc value gets put into the
            buffer
        *usart1_tdr = 'n ';
    }
}
```

Listing 4: ADC-Interrupt-Handler-Code

7 NVIC

7.1 Wir wird er verwendet

Beim Programmieren des STM muss der Vector Table beschrieben werden mit einem Assembler File und Linker Skript, diese werden von der Cube IDE erstellt.
Im Vector Table sind alle Interrupts mit dem entsprechenden Funktionnamen für das Programm hinterlegt. Im NVIC_ISER Register können die Interrupts durch ihre Interrupt Position aktiviert werden.

Table 32. Vector table (continued)

Position	Priority	Type of priority	Acronym	Description	Address
3	10	settable	FLASH	Flash global interrupt	0x0000 004C
4	11	settable	RCC	RCC global interrupts	0x0000 0050
5	12	settable	EXTI0_1	EXTI Line[1:0] interrupts	0x0000 0054
6	13	settable	EXTI2_3	EXTI Line[3:2] interrupts	0x0000 0058
7	14	settable	EXTI4_15	EXTI Line[15:4] interrupts	0x0000 005C
8			Reserved		0x0000 0060
9	16	settable	DMA_CH1	DMA channel 1 interrupt	0x0000 0064
10	17	settable	DMA_CH2_3	DMA channel 2 and 3 interrupts	0x0000 0068
11	18	settable	DMA_CH4_5	DMA channel 4 and 5 interrupts	0x0000 006C
12	19	settable	ADC	ADC interrupts	0x0000 0070
13	20	settable	TIM1_BRK_UP_TRG_COM	TIM1 break, update, trigger and commutation interrupt	0x0000 0074
14	21	settable	TIM1_CC	TIM1 capture compare interrupt	0x0000 0078
15			Reserved		0x0000 007C
16	23	settable	TIM3	TIM3 global interrupt	0x0000 0080
17	24	settable	TIM6	TIM6 global interrupt	0x0000 0084
18			Reserved		0x0000 0084
19			Reserved		0x0000 0088
19	26	settable	TIM14	TIM14 global interrupt	0x0000 008C
20	27	settable	TIM15	TIM15 global interrupt	0x0000 0090
21	28	settable	TIM16	TIM16 global interrupt	0x0000 0094
22	29	settable	TIM17	TIM17 global interrupt	0x0000 0098
23	30	settable	I2C1	I^2C1 global interrupt	0x0000 009C
24	31	settable	I2C2	I^2C2 global interrupt	0x0000 00A0
25	32	settable	SPI1	SPI1 global interrupt	0x0000 00A4
26	33	settable	SPI2	SPI2 global interrupt	0x0000 00A8
27	34	settable	USART1	USART1 global interrupt	0x0000 00AC
28	35	settable	USART2	USART2 global interrupt	0x0000 00B0
29	36	settable	USART3_4_5_6	USART3, USART4, USART5, USART6 global interrupts	0x0000 00B4
30			Reserved		0x0000 00B8
31	38	settable	USB	USB global interrupt (combined with EXTI line 18)	0x0000 00BC

Abbildung 11: NVIC-Vector-Table

4.2.2 Interrupt set-enable register (ISER)

Address offset: 0x00

Reset value: 0x0000 0000

The ISER register enables interrupts, and shows which interrupts are enabled

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

Bits 31:0 **SETENA**: Interrupt set-enable bits.
Write:
0: No effect
1: Enable interrupt
Read:
0: Interrupt disabled
1: Interrupt enabled.
If a pending interrupt is enabled, the NVIC activates the interrupt based on its priority. If an interrupt is not enabled, asserting its interrupt signal changes the interrupt state to pending, but the NVIC never activates the interrupt, regardless of its priority.

Abbildung 12: NVIC-ISER-Register

7.2 NVIC-Code

```
void NVIC_enable_interrupts(void)
{
    uint32_t* nvic_iser =  NVIC_ISER;
    uint32_t REG_CONTENT;
    REG_CONTENT = *nvic_iser;
    REG_CONTENT |= 0x08000800;
    *nvic_iser = REG_CONTENT;
}
```

Listing 5: NVIC-enable-interrupts

8 Gesamtes-Programm

8.1 Headerfile

```
#ifndef __header_H
#define __header_H

typedef unsigned        uint32_t;
typedef unsigned short  uint16_t;
typedef unsigned char   uint8_t;


//boundary addresses at page 37 - 41
/*
General boundaries
*/
#define PERIPHALS       ((uint32_t*)0x40000000)     //Peripherals
#define APBPERIPHALS    PERIPHALS                   //APBPeriphals
#define AHBPERIPHALS    (PERIPHALS + 0x00020000)
#define AHB2PERIPHALS   (PERIPHALS + 0x08000000)

/*
USART1
*/
#define USART1_BASE (APBPERIPHALS+0x00013800) //USART 1 Base Address
//Base + Address offset + Comment + Page in reference Manual
#define USART1_CR1      (USART1_BASE+0x00)   //USART Control Register 1  --
   at Page 625
#define USART1_CR2      (USART1_BASE+0x04)   //USART Control Register 2  --
   at Page 628
#define USART1_CR3      (USART1_BASE+0x08)   //USART Control Register 3  --
   at Page 630
#define USART1_BRR      (USART1_BASE+0x0C)   //USART Baudrate Register   --
   at page 632
#define USART1_RQR      (USART1_BASE+0x18)   //USART Request register

#define USART1_ISR      (USART1_BASE+0x1c)   //USART Interrupt and status
   register   -- at Page 635
#define USART1_ISR_TXE  ((uint32_t*)0x00000080)
#define USART1_ISR_TC   ((uint32_t*)0x00000040)
#define USART1_ISR_RXNE ((uint32_t*)0x00000020)

#define USART1_ICR       (USART1_BASE+0x20)   //USART Interrupt and flag
   Clear register -- at Page 638
#define USART1_ICR_TCCF  ((uint32_t*) 0x00000040) //USART transmission
   complete clear flag -- at Page 639
#define USART1_RDR       (USART1_BASE+0x24)   //USART Receive Data register
     --  at Page 639
#define USART1_TDR       (USART1_BASE+0x28)   //USART Transmit Data register
     --  at Page 640

/*
```

```
RCC
*/
//From boundary addresses at page 39
#define RCC (AHBPERIPHALS+0x00001000)
//All found from RCC Register Map on Page 125
#define RCC_CR          (RCC+0x00)          //RCC Control register -- at
   Page 99
#define RCC_CFGR        (RCC+0x04)          //RCC Clock configuration
   register -- at Page 101
#define RCC_CIR         (RCC+0x08)          //RCC Clock interrupt register
   -- at Page 104
#define RCC_APB2RSTR    (RCC+0x0C)          //RCC APB peripheral reset
   register 2 -- at Page 106
#define RCC_APB1RSTR    (RCC+0x010)         //RCC APB peripheral reset
   register 1 -- at Page 108
#define RCC_AHBENR      (RCC+0x14)          //RCC AHB peripheral clock
   enable register -- at Page 111
#define RCC_APB2ENR     (RCC+0x18)          //RCC APB peripheral clock
   enable register 2-- at Page 112
#define RCC_APB1ENR     (RCC+0x1C)          //RCC APB peripheral clock
   enable register 1-- at Page 114
#define RCC_BDCR        (RCC+0x20)          //RCC RTC domain control
   register -- at Page 117
#define RCC_CSR         (RCC+0x24)          //RCC Control/status register --
    at Page 119
#define RCC_AHBRSTR     (RCC+0x28)          //RCC AHB peripheral reset
   register -- at Page 120
#define RCC_CFGR2       (RCC+0x2C)          //RCC Clock configuration
   register 2 -- at Page 122
#define RCC_CFGR3       (RCC+0x30)          //RCC Clock configuration
   register 3 -- at Page 123
#define RCC_CR2         (RCC+0x34)          //RCC Clock control register 2
   -- at Page 123


/*
GPIOA
*/
#define GPIO_A (AHB2PERIPHALS + 0x00000000)
//Page 142
#define GPIOA_MODER     (GPIO_A+0x00)       //GPIO port moder register -- at
    Page 136
#define GPIOA_OTYPER    (GPIO_A+0x04)       //GPIO port output type register
    -- at Page 136
#define GPIOA_OSPEEDR   (GPIO_A+0x08)       //GPIO port output speed
   register -- at Page 137
#define GPIOA_PUPDR     (GPIO_A+0x0C)       //GPIO port pull-up/pull-down
   register -- at Page 137
#define GPIOA_IDR       (GPIO_A+0x10)       //GPIO port input data register
   -- at Page 138
#define GPIOA_ODR       (GPIO_A+0x14)       //GPIO port output data register
    -- at Page 138
```

```
#define GPIOA_BSRR      (GPIO_A+0x18)       //GPIO port bis set/reset
   register -- at Page 138
#define GPIOA_LCKR      (GPIO_A+0x1c)       //GPIO port configuration lock
   register -- at Page 139
#define GPIOA_AFRL      (GPIO_A+0x20)       //GPIO alternate function low
   register -- at Page 140
#define GPIOA_AFRH      (GPIO_A+0x024)      //GPIO alternate function high
   register -- at Page 141
#define GPIOA_BRR       (GPIO_A+0x28)       //GPIO port bit reset register
   -- at Page 141


/*
ADC
*/
//boundary address at page 40
#define ADC (APBPERIPHALS+0x00012400)
//Page 220
#define ADC_ISR         (ADC+0x00)                    //ADC interrupt and
   status register -- at Page 207
#define ADC_ISR_AWD     ((uint32_t*)0x00000080)       //Analog watchdog
   flag
#define ADC_ISR_OVR     ((uint32_t*)0x00000010)       //Overrun flag
#define ADC_ISR_EOSEQ   ((uint32_t*)0x00000008)       //End of Sequence
   flag
#define ADC_ISR_EOC     ((uint32_t*)0x00000004)       //End of Conversion
#define ADC_ISR_EOSMP   ((uint32_t*)0x00000002)       //End of sampling
   flag
#define ADC_ISR_ADRDY   ((uint32_t*)0x00000001)       //ADC Ready

#define ADC_IER         (ADC+0x04)      //ADC interrupt enable register --
   at Page 208

#define ADC_CR          (ADC+0x08)      //ADC control register -- at Page
   210
#define ADC_CR_ADCAL    ((uint32_t)0x80000000)  //ADC Calibration
#define ADC_CR_ADSTP    ((uint32_t)0x00000010)  //ADC stop of conversion
   command
#define ADC_CR_ADSTART  ((uint32_t)0x00000004)  //ADC start of conversion
#define ADC_CR_ADDIS    ((uint32_t)0x00000002)  //ADC disable command
#define ADC_CR_ADEN      ((uint32_t)0x00000001) //ADC enable control


#define ADC_CFGR1       (ADC+0x0C)      //ADC configuration register 1 -- at
    Page 212
#define ADC_CFGR2       (ADC+0x10)      //ADC configuration register 2 -- at
    Page 216
#define ADC_SMPR        (ADC+0x14)      //ADC sampling time register -- at
   Page 216
#define ADC_TR          (ADC+0x20)      //ADC watchdog threshold register --
    at Page 217
#define ADC_CHSELR      (ADC+0x28)      //ADC channel selection register --
   at Page 218
```

```
#define ADC_DR          (ADC+0x40)      //ADC data register -- at Page 218
#define ADC_DR_DATA    ((uint32_t)0x0000FFFF)        //data

#define ADC_CCR         (ADC+0x308)     //ADC common configuration register
   -- at Page 219




//Interrupt ant Page 171
#define ADC_IRQn    12          //Address 0x0000 0070
#define USART1_IRQn 27      //Address 0x0000 00AC
//in ARMv6 and stm32f0xx-cortexm0-programming-manual-- at Page 70
#define NVIC_ISER   ((uint32_t*)0xE000E100)   //Interrupt Set-Enable
   Register page B3-284
#define NVIC_ICER   ((uint32_t*)0xE000E180)   //Interrupt Clear Enable
   Register page B3-285
#define NVIC_ISPR   ((uint32_t*)0xE000E200)   //Interrupt Set-Pending
   Register page B3-286
#define NVIC_ICPR   ((uint32_t*)0xE000E280)   //Interrupt Clear-Pending
   Register page B3-287
#define NVIC_IPRn   ((uint32_t*)0xE000E400)   //-0xE000E43C Interrupt
   Priority Registers NVIC_IPR0-NVICIPR7 B3-288







#endif
```

Listing 6: Headerfile

8.2 Main

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include <unistd.h>
#include <fcntl.h>
#include "header.h"



#define USART1_interrupt ((uint32_t)0x000000AC)

char USART_READ;
uint32_t ADC_Value;

//Prototypes
char* USART_write_data;

//ADC Interrupt Handler
void ADC1_IRQHandler(void)
{
    uint32_t* adc_isr = ADC_ISR;
    uint32_t* adc_cr = ADC_CR;

    uint32_t* adc_isr_eoc = ADC_ISR_EOC;
    uint32_t* adc_isr_adrdy = ADC_ISR_ADRDY;
    uint32_t* adc_dr = ADC_DR;
    uint32_t* adc_dr_data = ADC_DR_DATA;
    uint32_t* gpioa_odr = GPIOA_ODR;
    uint16_t* usart1_tdr =  USART1_TDR;

    uint32_t adc_cr_adstart = ADC_CR_ADSTART;
    uint32_t ADC_Value;
    uint32_t REG_CONTENT;

    //if ADC Ready start convertion
    if((*adc_isr & *adc_isr_adrdy)== 1)
    {
        REG_CONTENT = *adc_cr;
        REG_CONTENT |= adc_cr_adstart;
        *adc_cr = REG_CONTENT;

    }
    //if convertion complete send ADC value
    if((*adc_isr & *adc_isr_eoc) == 1)
    {
        ADC_Value = (*adc_dr & *adc_dr_data);
        //Set PA7 and PA6 high for max to send data
        REG_CONTENT = *gpioa_odr;
        REG_CONTENT |= 0x000000C0;
        *gpioa_odr = REG_CONTENT;
```

```
        USART_write_data = (char*)ADC_Value;
        //write something into the USART Buffer for USART
        //interrupt where rest of adc value gets put into the buffer
        *usart1_tdr = 'n';
    }
}

//Enable Interrupts in NVIC
void NVIC_enable_interrupts(void)
{
    uint32_t* nvic_iser =  NVIC_ISER;
    uint32_t REG_CONTENT;
    REG_CONTENT = *nvic_iser;
    REG_CONTENT |= 0x08000800;
    *nvic_iser = REG_CONTENT;
}


//USART Interrupt Handler
void USART1_IRQHandler(void)
{
    uint32_t* usart1_isr = *USART1_ISR;
    uint32_t* usart1_isr_rxne = *USART1_ISR_RXNE;
    uint32_t* usart1_isr = USART1_ISR;
    uint32_t* usart1_isr_tc = USART1_ISR_TC;
    uint32_t* usart1_tc = USART1_ICR;
    uint32_t* usart1_tc_tcce = USART1_ICR_TCCF;
    uint16_t* usart1_tdr =  USART1_TDR;
    uint32_t* gpioa_odr = GPIOA_ODR;
    uint32_t REG_CONTENT;
    int send=0;

    if ((*usart1_isr & *usart1_isr_tc) == 1)
    {
        if (send == sizeof(USART_write_data))
        {
            //set MAX to listen
            REG_CONTENT = *gpioa_odr;
            REG_CONTENT |= 0x00000000;
            *gpioa_odr = REG_CONTENT;

            send=0;

            *usart1_tc |= *usart1_tc_tcce; /* Clear transfer complete flag
                */
        }
        else
        {
            /* clear transfer complete flag and fill TDR with a new char */
            *usart1_tdr = USART_write_data[send++];
        }
    }
```

```
    if ( (*usart1_isr & *usart1_isr_rxne) == *usart1_isr_rxne)
    {
        USART_READ = *((char *)USART1_RDR);
    }
}

//Initialize ADC
int init_ADC()
{
    uint32_t REG_CONTENT;
    uint32_t* adc_chselr = ADC_CHSELR;
    uint32_t* adc_isr = ADC_ISR;
    uint32_t* adc_cr = ADC_CR;
    uint32_t* adc_cr_adcal = ADC_CR_ADCAL;
    uint32_t* adc_ier = ADC_IER;
    uint32_t* adc_cfgr2 = ADC_CFGR2;
    uint32_t adc_isr_adrdy = ADC_ISR_ADRDY;


    //set ADC clock to PCLK so thats is synchronous with sysclock
    REG_CONTENT = *adc_cfgr2;
    REG_CONTENT |= 0x40000000;
    *adc_cfgr2 = REG_CONTENT;

    //before starting callibrate ADC
    REG_CONTENT = *adc_cr;
    REG_CONTENT |= 0x80000000;
    *adc_cr = REG_CONTENT;


    //if calibration is complete enable ADC
    //enable Interrupts
    //set correct channel
    //when ADC is ready, a ad ready interrupt occurs and the interrupt
       handler
    //will then start the ADC convertion
    if((*adc_cr & *adc_cr_adcal) == 0)
    {
        //enable ADC and ensure ADSTART=0 for further configuration
        REG_CONTENT = *adc_cr;
        REG_CONTENT |= 0x00000005;
        *adc_cr = REG_CONTENT;

        //enable Interrupts of ADC
        REG_CONTENT = *adc_ier;
        REG_CONTENT |= 0x00000005;
        *adc_ier = REG_CONTENT;

        //Set ADC Channel to channel 0 because PA0 is ADC_IN0
        REG_CONTENT = *adc_chselr;
        REG_CONTENT |= 0x00000001;
        *adc_chselr = REG_CONTENT;
```

```
    }

}

//Initialize needed clocks
int init_CLOCK()
{
    uint32_t REG_CONTENT;
    uint32_t* rcc_ahbenr = RCC_AHBENR;
    uint32_t* rcc_apb2enr = RCC_APB2ENR;
    uint32_t* rcc_cfgr = RCC_CFGR;

    REG_CONTENT = *rcc_ahbenr;
    REG_CONTENT |= 0x00020000;
    *rcc_ahbenr = REG_CONTENT;

    REG_CONTENT = *rcc_apb2enr;
    REG_CONTENT |= 0x00004000;
    *rcc_apb2enr = REG_CONTENT;

    //set AHB Clock to not divided so same clock as sysclock
    REG_CONTENT = *rcc_cfgr;
    REG_CONTENT |= 0x00000000;
    *rcc_cfgr = REG_CONTENT;
}

//Initialize GPIOs
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
    //PA0 ADC_IN0 so set it to analog and then in DAC register set PA0 as
        ADC_IN0
    uint32_t REG_CONTENT;
    uint32_t* gpioa_moder = GPIOA_MODER;
    uint32_t* gpioa_afrh = GPIOA_AFRH;
    uint32_t* gpioa_odr = GPIOA_ODR;

    REG_CONTENT = *gpioa_moder;
    REG_CONTENT |= 0x00285003;
    *gpioa_moder = REG_CONTENT;

    //Set GPIO PA9 as AF=TX and PA10 as AF = RX
    REG_CONTENT = *gpioa_afrh;
    REG_CONTENT |= 0x00000110;
    *gpioa_afrh = REG_CONTENT;

    //Ensure that output gpios are 0 for MAX, to read all the time
    //and only send, when needing to send
    REG_CONTENT = *gpioa_odr;
    REG_CONTENT |= 0x00000000;
    *gpioa_odr = REG_CONTENT;
}
```

```
//Initialize USART
int init_UART()
{
    uint32_t REG_CONTENT;
    uint32_t* rcc_cfr3 = RCC_CFGR3;
    uint32_t* usart1_brr = USART1_BRR;
    uint32_t* usart1_cr1 = USART1_CR1;

    //Use SYSCLK for USART and use Baudrate 38400
    REG_CONTENT = *rcc_cfr3;
    REG_CONTENT |= 0x00000001;
    *rcc_cfr3 = REG_CONTENT;

    REG_CONTENT = *usart1_brr;
    REG_CONTENT = 0x00000D00;
    *usart1_brr = REG_CONTENT;

    //Wordlenght, Parity control enable, Parity selection, interrupt enable,
        Transmission complete interrupt enable, RXNE interrupt enable
    REG_CONTENT = *usart1_cr1;
    REG_CONTENT |= 0x000006EC;
    *usart1_cr1 = REG_CONTENT;

    REG_CONTENT = *usart1_cr1;
    REG_CONTENT |= 0x00000001; //enable UART
    *usart1_cr1 = REG_CONTENT;
}

int main()
{
    init_CLOCK();
    init_GPIO();
    init_UART();
    init_ADC();
    NVIC_enable_interrupts();

    for(;;);
}
```

Listing 7: Gesamter-Code

9 Anhang

9.1 Verlinkungen

Abbildung: 0
`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