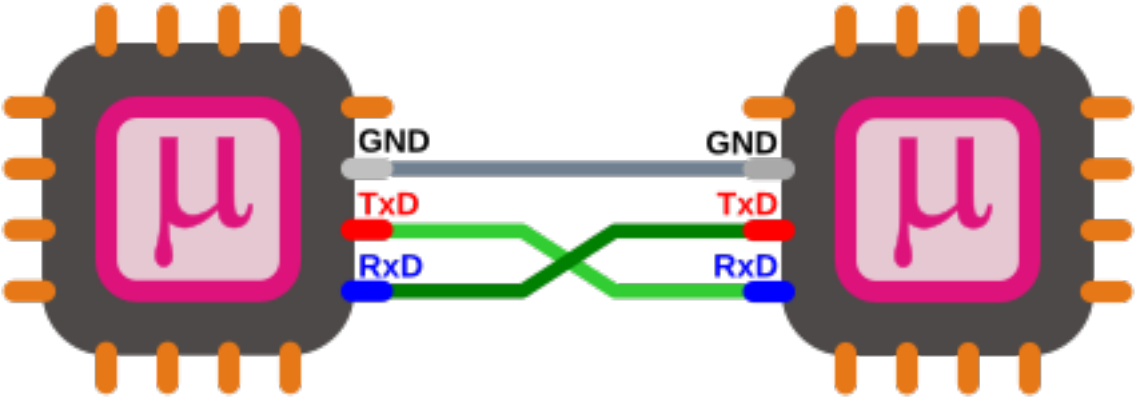
DIC-Serielle Kommunikation mit einem µC

Fabio Plunser

8. Dezember 2020



Inhaltsverzeichnis

Abbildungsverzeichnis

Code

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

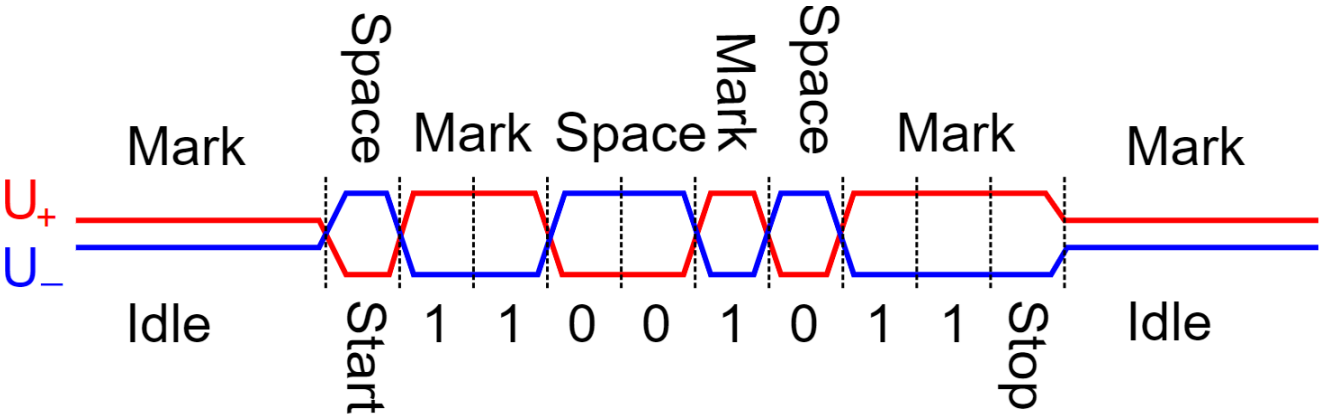


Abbildung 1: RS485-Diagramm

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.

Ebenso muss der

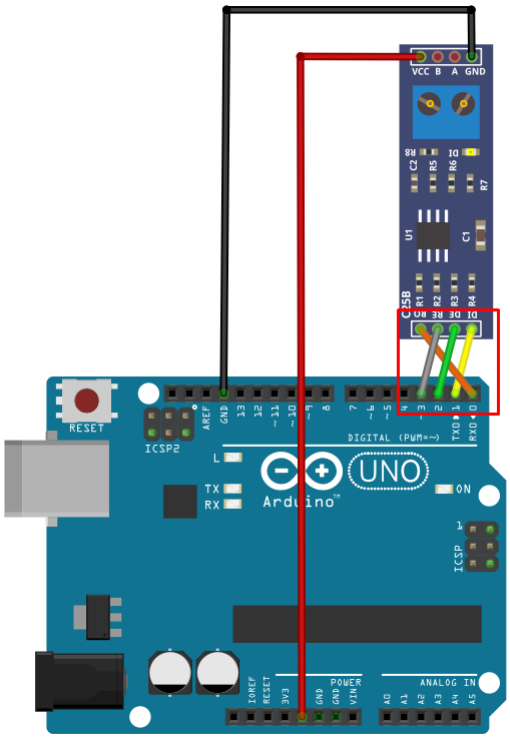


Abbildung 2: MAX485-Arduino-Anschluss-BSP

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.

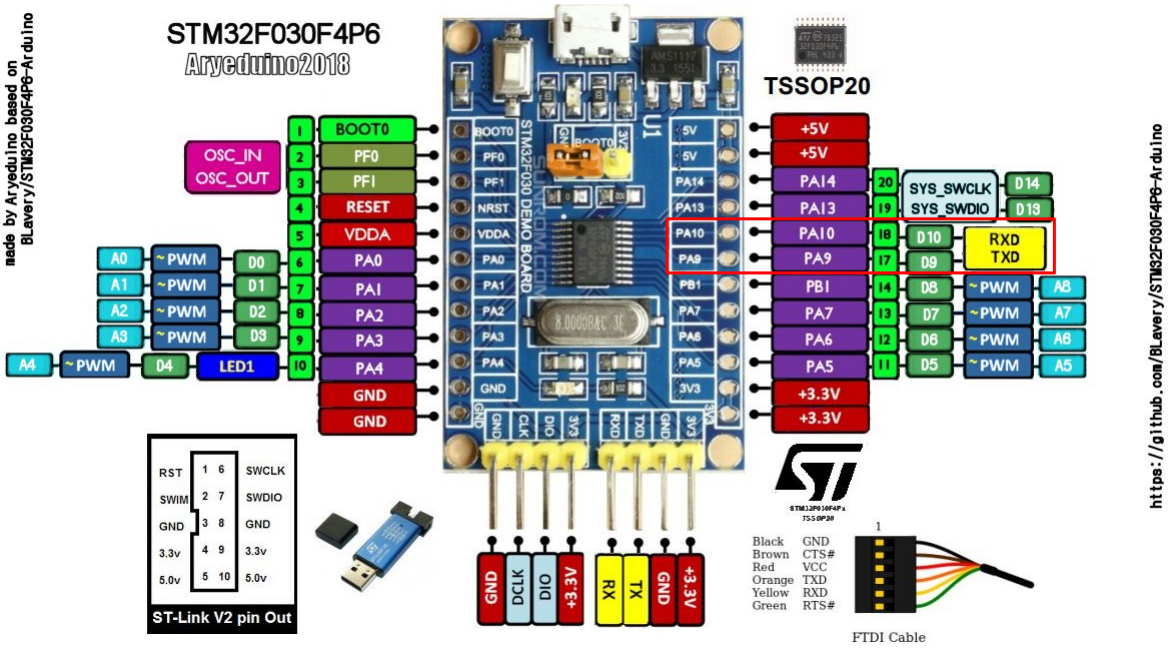


Abbildung 3: STM32F030F4P6-Pinout

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**

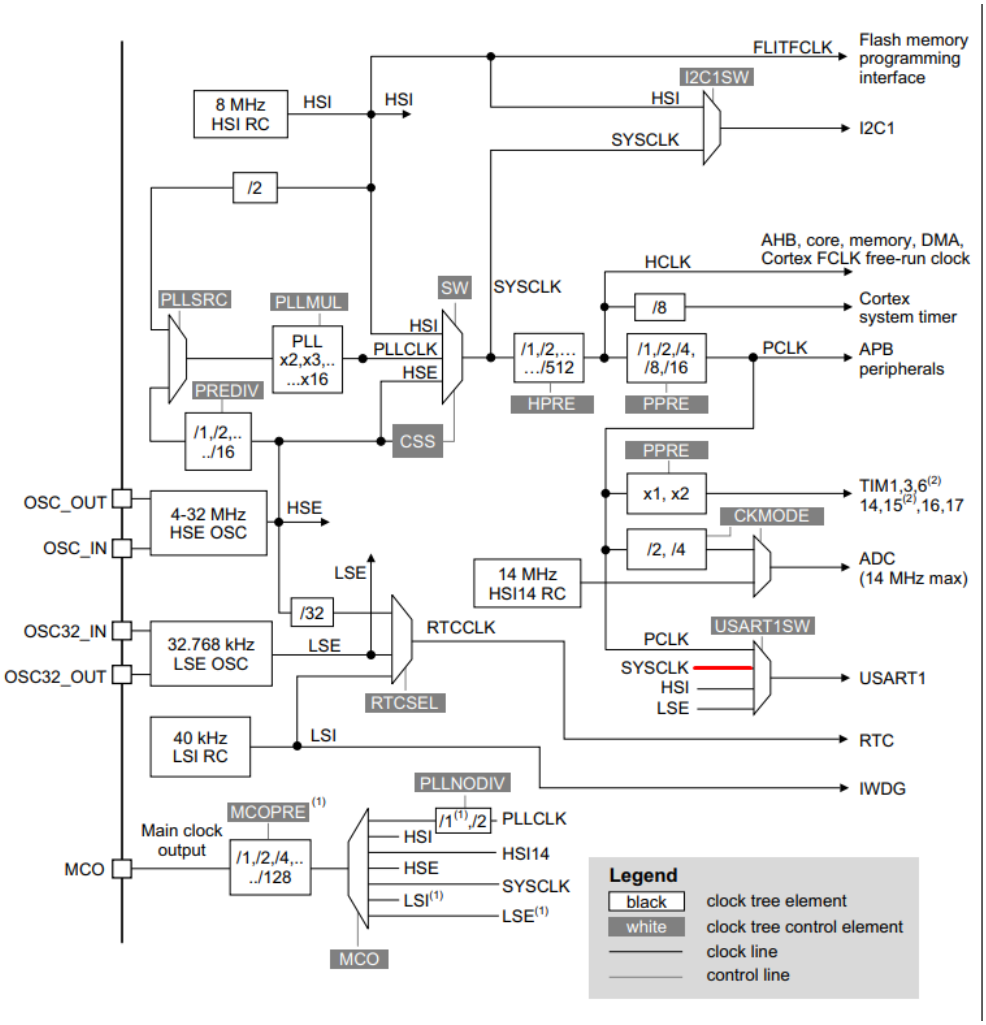


Abbildung 4: Clock-Tree

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 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.

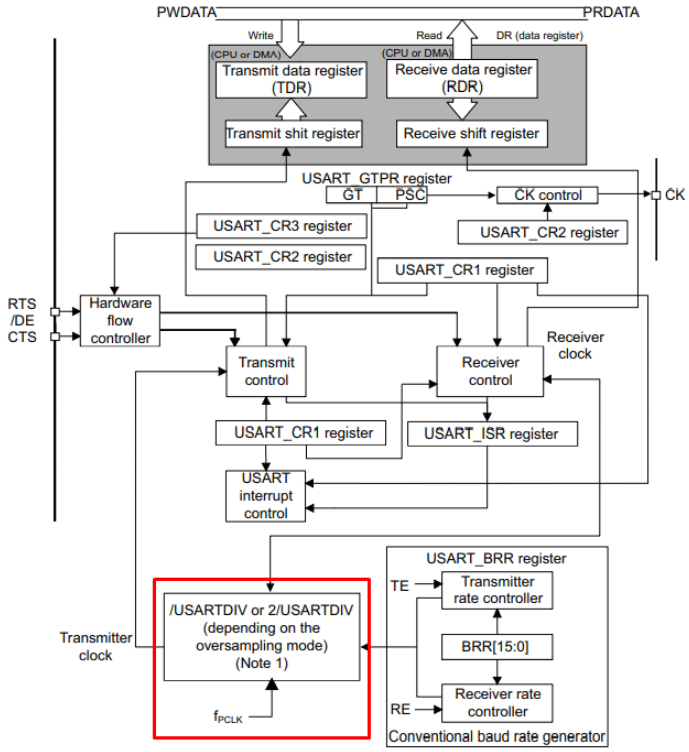


Abbildung 5: USART-Aufbaug

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 6: USART_BRR

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 = 0x00000D00; //=
       00000000000000000000110100000000
    *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 7 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 8, 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 9 und 10

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.

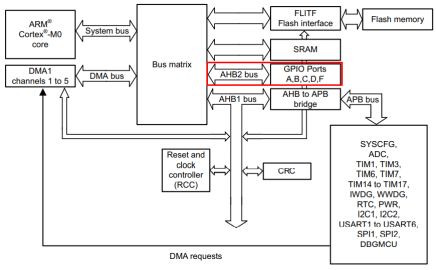


Abbildung 7: System-Architektur

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 8: 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) USART2_CTS(1)(3)	-	-	USART4_TX(1)	-	-
PA1	EVENTOUT	USART1_RTS(2) USART2_RTS(1)(3)	-	-	USART4_RX(1)	TIM15_CH1N(1)	-
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) USART2_CK(1)(3)	-	-	TIM14_CH1	USART6_TX(1)	-
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 9: 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	1000: Reserved
0001: AF1	1001: Reserved
0010: AF2	1010: Reserved
0011: AF3	1011: Reserved
0100: AF4	1100: Reserved
0101: AF5	1101: Reserved
0110: AF6	1110: Reserved
0111: AF7	1111: Reserved

Abbildung 10: 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

6.1 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*)0x48000000)
#define APBPERIPHALS    PERIPHALS
#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 GPIOx_MODER     (GPIO_A+0x00)       //GPIO port moder register -- at
    Page 136
#define GPIOx_OTYPER    (GPIO_A+0x04)       //GPIO port output type register
    -- at Page 136
#define GPIOx_OSPEEDR   (GPIO_A+0x08)       //GPIO port output speed
   register -- at Page 137
#define GPIOx_PUPDR     (GPIO_A+0x0C)       //GPIO port pull-up/pull-down
   register -- at Page 137
#define GPIOx_IDR       (GPIO_A+0x10)       //GPIO port input data register
   -- at Page 138
#define GPIOx_ODR       (GPIO_A+0x14)       //GPIO port output data register
    -- at Page 138
#define GPIOx_BSRR      (GPIO_A+0x18)       //GPIO port bis set/reset
   register -- at Page 138
```

```
#define GPIOx_LCKR      (GPIO_A+0x1c)       //GPIO port configuration lock
   register -- at Page 139
#define GPIOx_AFRL      (GPIO_A+0x20)       //GPIO alternate function low
   register -- at Page 140
#define GPIOx_AFRH      (GPIO_A+0x024)      //GPIO alternate function high
   register -- at Page 141
#define GPIOx_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_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_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 3: Headerfile

6.2 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)


/*
//ADC Interrupt at Page 206
//Generel Interrupts at Page 170
A.6.1 NVIC initialization example
(1) Enable Interrupt on ADC
(2) Set priority for ADC to 2
NVIC_EnableIRQ(ADC1_COMP_IRQn);  (1)
NVIC_SetPriority(ADC1_COMP_IRQn,2);  (2)
NVIC im ARMv6 s283

Page 174
*/

//Prototypes
int USART_Send (char* string);

//When ADC End of Conversion send Value
void ADC_IRQHandler(void)
{
    if(*ADC_ISR & *ADC_ISR_EOC == *ADC_ISR_EOC)
    {
        USART_Send("Value");
    }

}

void enable_usart_adc_interrupts(void)
{
    uint32_t* nvic_iser = (uint32_t*) NVIC_ISER;
    *nvic_iser |= 0x08001000;
}
int USART_Send(char* string)
{

    int send=0;
    if ((*USART1_ISR & *USART1_ISR_TC) == *USART1_ISR_TC)
    {
        if (send == sizeof(string))
```

```
        {
            send=0;
            *USART1_ICR |= *USART1_ICR_TCCF; /* Clear transfer complete flag
                */
            return EXIT_SUCCESS;
        }
        else
        {
            /* clear transfer complete flag and fill TDR with a new char */
            *USART1_TDR = string[send++];
        }
    }

    // uint32_t* usart1_isr = (uint32_t*) USART1_ISR;
    // uint32_t* usart1_isr_tc = (uint32_t*) USART1_ISR_TC;
    // uint32_t* usart1_tc = (uint32_t*) USART1_ICR;
    // uint32_t* usart1_tc_tcce = (uint32_t*) USART1_ICR_TCCF;
    // uint32_t* usart1_tdr = (uint32_t*) USART1_TDR;
    // int send=0;
    // if ((*usart1_isr & *usart1_isr_tc) == *usart1_isr_tc)
    // {
    //     if (send == sizeof(string))
    //     {
    //         send=0;
    //         *usart1_tc |= *usart1_tc_tcce; /* Clear transfer complete
    flag */
    //         return EXIT_SUCCESS;
    //     }
    //     else
    //     {
    //         /* clear transfer complete flag and fill TDR with a new char
    */
    //         *usart1_tdr = string[send++];
    //     }
    // }


    // uint32_t USART1_ISR_t = *USART1_ISR;
    // uint32_t USART1_FLAG_TXE = *USART1_ISR_TXE;


    // if (USART1_ISR_t == USART1_FLAG_TXE)
    // {
    //     *((uint16_t**)USART1_TDR) = (uint16_t*)Char;
    // }

}

char USART_Receive()
{
    uint32_t USART1_ISR_t = *USART1_ISR;
    uint32_t USART1_IT_RXNE = *USART1_ISR_RXNE;
    char readchar;
```

```
    if ( USART1_ISR_t & USART1_IT_RXNE == USART1_IT_RXNE)
    {
        readchar = *((char *)USART1_RDR);
    }
    return readchar;
}


int init_CLOCK()
{
    uint32_t REG_CONTENT;
    REG_CONTENT = *RCC_AHBENR;
    REG_CONTENT |= 0x00020000;
    *RCC_AHBENR = REG_CONTENT;

    REG_CONTENT = *RCC_APB2ENR;
    REG_CONTENT |= 0x00004000;
    *RCC_APB2ENR = REG_CONTENT;
}
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
    uint32_t REG_CONTENT;
    REG_CONTENT = *GPIOx_MODER;
    REG_CONTENT |= 0x00285000;
    *GPIOx_MODER = REG_CONTENT;

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

int init_UART()
{

    uint32_t REG_CONTENT;

    //Use SYSCLK for USART and use Baudrate 38400
    REG_CONTENT = *RCC_CFGR3;
    REG_CONTENT |= 0x00000001;
    *RCC_CFGR3 = 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();


    for(;;)
    {

    }
}
```

Listing 4: 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