DIC Projekt 1

Serielle Kommunikation mit einem uC

Autor: Patrick Wintner, 5BHEL
Betreuer: Lezuo

Abgabedatum: 27.02.2021

Contents

Chapter 1

Allgmeines

1.1 Aufgabenstellung

Ein 10 Byte großer Buffer, welcher Messdaten des ADC eines STM32F030F-uCs enthält, soll über RS485 per USART3 übertragen werden, wenn Daten über USART3 empfangen werden. Ein MAX485 ist auf die Pins PC10 und PC11 geroutet, weiters werden eine Baud-Rate von 38400 und ungerade Parität verwendet.

1.2 RS485

RS485 ist ein Industriestandard für eine Schnittstelle für serielle Datenkommunikation, welcher eine symmetrische Leitung nutzt.
RS-485 nutzt ein Leiterpaar, wobei einer diese Leitungen invertiert und die andere nicht invertiert ist, um den Pegel eines 1-Bit-Signals zu übertragen. Der Empfänger wertet dann nur die Differenz aus, was den Vorteil hat, dass Gleichtaktstörungen (also ein unerwünschter Offset auf den Leitungen) keinen Einfluss mehr auf die Übertragung haben. Weiters sind nur die elektrischen Spezifikationen definiert, weshalb das Übertragungsprotokoll gewählt werden kann, was dazu führt, das Bausteine mehrerer Hersteller oftmals nicht kompatibel zueinander sind.
Senderseitig betragen die Signalspannungen mindestens +/-1,5V und höchstens +/-6V. Der Treiber wird oft als Brückenschaltung ausgeführt.
Empfangsseitig muss die Umschaltschwelle im Bereich von +/-1,2V liegen.
RS485 wird meistens im half-duplex betrieben (würden zwei Aderpaare verwendet werden, könnte ein Full-Duplex-Betrieb stattfinden) und ist multipointfähig, was bedeutet, dass mehrere Teilnehmer an einem Bus angeschlossen

werden können. Die Maximalanzahl an Teilnehmern, welche 32 beträgt, kann erhöht werden, indem Teilnehmer mit geringerer "Unit Load" verwendet werden, wodurch die Teilnehmerzahl bei entsrechend geringer Unit Load auf 256 erhöht werden kann.
Da RS485 ein Bussystem ist, sollten an den Enden der Leitung Terminierungswiderstände von je 120Ohm angeschlossen werden. Optional können diese zu Bias-Netzwerken ausgebaut werden, um den Störabstand bei inaktiven Treibern zu verbessern.

Spezifikationen:

- Anzahl Empfänger: 32
- Maximale Leitungslänge: 1200m
- Maximale Datenübertragungsrate: 120Mb/s
- Gleichtakt-Eingangsspannung: -7 bis +12V
- Eingangswiderstand des Empfängers: 12kOhm (1 Unit Load)
- Eingangsempfindlichkeit des Empfängers: +/-200mV

Transmitter				Receiver			
DI	**DE**	**Y**	**Z**	**A − B**	**$\overline{RE}$**	**RO**
H	H	H	L	$U_{ID} \geq 0{,}2$ V	L	H
L	H	L	H	$0{,}2\ V > U_{ID} > -0{,}2$ V	L	n. def.
x	L	Z	Z	$U_{ID} \leq -0{,}2$ V	L	L
				x	H	Z
				Open	L	n. def.

Transmitter:
DI ... drive input
DE ... drive enable input
Y ... n. inv. driver output
Z ... inv. driver output
Receiver
A ... receiver input for Y
B ... receiver input for Z
/RE ... receiver enable (active Low)
RO ... receiver output

Kompatibilität zu RS-422: RS-485-Bauteile können bei RS-422-Nethwerken verwendet werden, umgekehrt nicht immer.

1.3 MAX485

1.3.1 Allgemeines

Der MAX485 ist ein Transceiver (kann sowohl Daten senden als auch empfangen), welcher für den RS485-Standard entwickelt wurde. Er verfügt über folgende Eigenschaften:

Auszüge aus dem Datenblatt des MAX485 S. 1-3

PART NUMBER	HALF/FULL DUPLEX	DATA RATE (Mbps)	SLEW-RATE LIMITED	LOW-POWER SHUTDOWN	RECEIVER/ DRIVER ENABLE	QUIESCENT CURRENT (µA)	NUMBER OF RECEIVERS ON BUS	PIN COUNT
MAX485	Half	2.5	No	No	Yes	300	32	8

DC ELECTRICAL CHARACTERISTICS

(V_{CC} = 5V ±5%, T_A = T_{MIN} to T_{MAX}, unless otherwise noted.) (Notes 1, 2)

PARAMETER	SYMBOL	CONDITIONS	MIN	TYP	MAX	UNITS
Input High Voltage	V_{IH}	DE, DI, $\overline{RE}$	2.0			V
Input Low Voltage	V_{IL}	DE, DI, $\overline{RE}$			0.8	V
Receiver Output High Voltage	V_{OH}	I_O = -4mA, V_{ID} = 200mV	3.5			V
Receiver Output Low Voltage	V_{OL}	I_O = 4mA, V_{ID} = -200mV			0.4	V

SWITCHING CHARACTERISTICS—MAX481/MAX485, MAX490/MAX491, MAX1487

(V_{CC} = 5V ±5%, T_A = T_{MIN} to T_{MAX}, unless otherwise noted.) (Notes 1, 2)

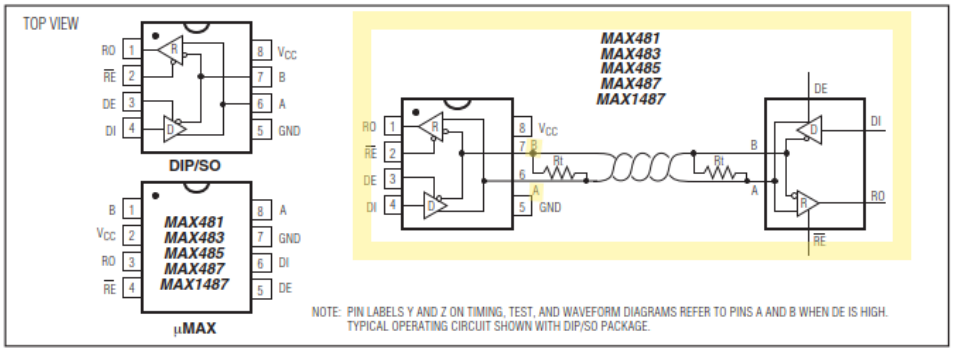
PARAMETER	SYMBOL	CONDITIONS		MIN	TYP	MAX	UNITS
Driver Rise or Fall Time	t_R, t_F	Figures 6 and 8, R_{DIFF} = 54Ω, C_{L1} = C_{L2} = 100pF	MAX481, MAX485, MAX1487	3	15	40	ns
			MAX490C/E, MAX491C/E	5	15	25	
			MAX490M, MAX491M	3	15	40	
Driver Enable to Output High	t_{ZH}	Figures 7 and 9, C_L = 100pF, S2 closed			40	70	ns
Driver Enable to Output Low	t_{ZL}	Figures 7 and 9, C_L = 100pF, S1 closed			40	70	ns
Driver Disable Time from Low	t_{LZ}	Figures 7 and 9, C_L = 15pF, S1 closed			40	70	ns
Driver Disable Time from High	t_{HZ}	Figures 7 and 9, C_L = 15pF, S2 closed			40	70	ns

Die Umschaltzeiten spielen bei der Konfiguration der USART später eine Rolle.

1.3.2 Pin-Beschreibung

Auszug aus Datenblatt des MAX48S. 7

PIN					NAME	FUNCTION
MAX481/MAX483/ MAX485/MAX487/ MAX1487		MAX488/ MAX490		MAX489/ MAX491		
DIP/SO	µMAX	DIP/SO	µMAX	DIP/SO		
1	3	2	4	2	RO	Receiver Output: If A > B by 200mV, RO will be high; If A < B by 200mV, RO will be low.
2	4	—	—	3	$\overline{RE}$	Receiver Output Enable. RO is enabled when $\overline{RE}$ is low; RO is high impedance when $\overline{RE}$ is high.
3	5	—	—	4	DE	Driver Output Enable. The driver outputs, Y and Z, are enabled by bringing DE high. They are high impedance when DE is low. If the driver outputs are enabled, the parts function as line drivers. While they are high impedance, they function as line receivers if $\overline{RE}$ is low.
4	6	3	5	5	DI	Driver Input. A low on DI forces output Y low and output Z high. Similarly, a high on DI forces output Y high and output Z low.
5	7	4	6	6, 7	GND	Ground
—	—	5	7	9	Y	Noninverting Driver Output
—	—	6	8	10	Z	Inverting Driver Output
6	8	—	—	—	A	Noninverting Receiver Input and Noninverting Driver Output
—	—	8	2	12	A	Noninverting Receiver Input
7	1	—	—	—	B	Inverting Receiver Input and Inverting Driver Output
—	—	7	1	11	B	Inverting Receiver Input
8	2	1	3	14	VCC	Positive Supply: 4.75V ≤ VCC ≤ 5.25V
—	—	—	—	1, 8, 13	N.C.	No Connect—not internally connected



NOTE: PIN LABELS Y AND Z ON TIMING, TEST, AND WAVEFORM DIAGRAMS REFER TO PINS A AND B WHEN DE IS HIGH. TYPICAL OPERATING CIRCUIT SHOWN WITH DIP/SO PACKAGE.

Die Bezeichnungen der Ein- und Ausgänge stimmen weitgehend mit denen des weiter oben erläuterten Standardbezeichnungen für den RS485-Standard. Der einzige Unterschied besteht in der Doppelfunktion der Sendeaus- und Empfangseingänge A und B - diese übernehmen nicht durch den Empfang von Daten (wie weiter oben), sondern dienen auch als Sendeausgänge, wobei A die Funktion von Y und B die Funktion von Z übernimmt. Es erscheint sinnvoll, die Pins DE und /RE mit der gleichen Leitung anzusteuern, weil damit entweder der Baustein als Empfänger (DE=RE=0) oder als Sender fungiert (DE=RE=1).

1.3.3 Wahrheitstabellen

Auszug aus Datenblatt des MAX48S. 10

Function Tables (MAX481/MAX483/MAX485/MAX487/MAX1487)

Table 1. Transmitting

INPUTS			OUTPUTS	
$\overline{RE}$	DE	DI	Z	Y
X	1	1	0	1
X	1	0	1	0
0	0	X	High-Z	High-Z
1	0	X	High-Z*	High-Z*

X = Don't care
High-Z = High impedance
**Shutdown mode for MAX481/MAX483/MAX487*

Table 2. Receiving

INPUTS			OUTPUT
$\overline{RE}$	DE	A-B	RO
0	0	≥ +0.2V	1
0	0	≤ -0.2V	0
0	0	Inputs open	1
1	0	X	High-Z*

X = Don't care
High-Z = High impedance
**Shutdown mode for MAX481/MAX483/MAX487*

Diese stimmen mit denen des RS485-Standards überein, mit Ausnahme der bereits erwähnten Doppelfunktion der Empfangsein- und Sendeausgänge A und B (A ersetzt Y, B ersetzt Z).

1.4 Pegel

Logikpegel dienen der Repräsentation von den logischen Werten 0 und 1 in der Digitaltechnik. Diese sind oft Spannungen. Es wird zwischen Low-Pegeln (L-Pegeln, Low, L) und High-Pegeln (H-Pegeln, High, H) unterschieden. Bei Spezifikationen werden für L Maximal- und für H Minimalpegel vorgegeben:

V_{IL} ... maximale Eingangsspannung, bei welcher noch arantiert wird, dass diese als L interpretiert wird
V_{IH} ... minimale Eingangsspannung, bei welcher noch garantiert wird, dass diese als H interpretiert wird
V_{OL} ... maximale Ausgangsspannung für L
V_{OH} ... minimale Ausgangsspannung für H

Zusätzlich gilt, dass $V_{IL} > V_{OL}$ und dass $V_{IH} < V_{OH}$. Dadurch soll sichergestellt werden, dass Eingänge die Werte der ihnen angeschlossenen Ausgänge richtig interpretieren können.

1.5 Half-Duplex

Wenn Kommunikation half-duplex erfolgt, so werden Daten abwechselnd, aber nicht gleichzeitig, in beide Richtungen gesandt werden. Daraus folgt, dass die kommunizierenden Geräte sich an ein Regelwerk halten müssen, um sich nicht gegenseitig zu behindern. Da der STM32 dieses Projektes laut Aufgabenstellung eine reaktionäre Rolle spielt (Er sendet Daten nur, nachdem er welche empfangen hat), wird davon ausgegangen, dass die Gegenseite nach dem Senden des den Sendeprozess auslösenden Zeichens solange mit weiteren Übertragungen wartet, bis die Übertragung des 10-Byte-Buffers abgeschlossen ist.

Chapter 2

Konfiguration des STM32F030F4

2.1 Einführung

Beim STM32F030F4-Chip handelt es sich um einen auf ARMv6-M-basierten Microkontroller, welcher mit maximal 48MHz arbeitet, 16kB Flash-Speicher und 4kB SRAM aufweist und eine Reihe von Peripheriegeräten, darunter USART-Schnittstellen und einen ADC enthält. Aus der Tabelle 2 des Datenblattes auf Seite 10 geht hervor, dass Chips der STM32F030xF-Serie lediglich über die USART1-Schnittstelle verfügen, weshalb, entgegen der Aufgabenstellung, USART1 anstatt USART3 verwendet wird und deshalb auch nicht die Pins PC10 und PC11 zum Anschluss des MAX485 verwendet werden.

Table 2. STM32F030x4/x6/x8/xC family device features and peripheral counts

Peripheral		STM32 F030F4	STM32 F030K6	STM32 F030C6	STM32 F030C8	STM32 F030CC	STM32 F030R8	STM32 F030RC
Flash (Kbytes)		16	32	32	64	256	64	256
SRAM (Kbytes)			4		8	32	8	32
Timers	Advanced control		1 (16-bit)					
	General purpose		4 (16-bit)(1)		5 (16-bit)			
	Basic		-		1 (16-bit)(2)	2 (16-bit)	1 (16-bit)(2)	2 (16-bit)
Comm. interfaces	SPI		1(3)		2			
	I^2C		1(4)		2			
	USART		1(5)		2(6)	6	2(6)	6
12-bit ADC (number of channels)		1 (9 ext. +2 int.)	1 (10 ext. +2 int.)	1 (10 ext. +2 int.)	1 (10 ext. +2 int.)	1 (10 ext. +2 int.)	1 (16 ext. +2 int.)	1 (16 ext. +2 int.)
GPIOs		15	26	39	39	37	55	51
Max. CPU frequency			48 MHz					
Operating voltage			2.4 to 3.6 V					
Operating temperature			Ambient operating temperature: -40°C to 85°C Junction temperature: -40°C to 105°C					
Packages		TSSOP20	LQFP32	LQFP48			LQFP64	

1. TIM15 is not present.
2. TIM7 is not present.
3. SPI2 is not present.
4. I2C2 is not present.
5. USART2 to USART6 are not present.
6. USART3 to USART6 are not present

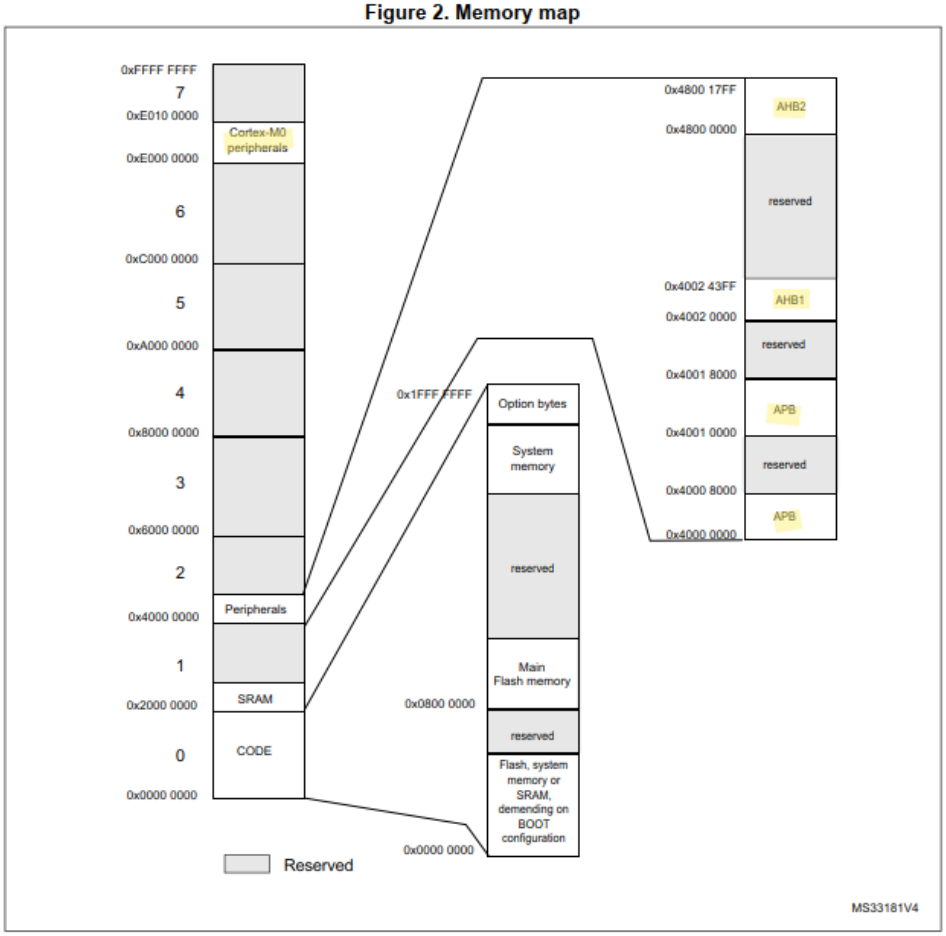
In diesem Dokument werden oft Auszüge aus verschiedenen Datenblättern und Reference-Manuals verwendet, dabei wird deren Quelle und die Seitenzahl angegeben. Bei der Quellenangabe wird dabei oft das STM-Reference Manual als RM abgekürzt.

2.2 Memory Organization

siehe RM S. 36ff

Sämtliche Speicher (Datenspeicher, Programmspeicher, Register, etc.) befinden sich im selben linearen 4GB-Adressraum, welcher in 8 Bereiche zu je 512MB aufgeteilt ist:

Figure 2. Memory map



Der für dieses Projekt wichtigste Speicherbereich ist jener zum Ansprechen und zur Konfiguration der Peripherie. Im STM-RM S. 37-41 in der Tabelle 1 werden die Grenzadressen der verschiedenen Speicherbereiche aufgelistet. Folgende sind zur Erfüllung der Aufgabenstellung von Belang:

- Port A: 0x4800 0000 - 0x4800 03FF
- RCC: 0x4002 1000 - 0x4002 13FF
- USART1: 0x4001 3800 - 0x4001 3BFF
- ADC: 0x4001 2400 - 0x4001 27FF

Anmerkung: Bei der Konfiguration der Register sind vor allem die Star-

tadressen wichtig, weil im Reference Manual der Address Offset des jeweiligen Registers angegeben ist.

2.3 Boot-Konfiguration

siehe RM S. 43

Der STM32F030F4 kann entweder vom Flash Memory, System Memory oder Embedded RAM gebootet werden. Die Auswahl erfolgt durch das Setzen des nBOOT1 Bit im User Option Byte und durch den logischen Wert am Pin BOOT0. Sofern Letzterer auf Masse gezogen wird, wird unabhängig vom Ersteren immer vom Main Flash gestartet; da entschieden wurde, dass der Bootvorgang vom Flash-Speicher ausgehen soll, ist keine weitere Konfiguration durch Software notwendig.

2.4 Pin-Konfiguration

siehe Datenblatt STM32F030F4 S. 25ff und RM S. 127ff

2.4.1 Allgemeines

Die Konfiguration der GPIO-Pins erfolgt über 32bit große GPIO-Register. Laut RM verfügt jeder Pin über vier Konfigurationsregister, zwei Datenregister und ein Set/Reset-Register, Pins der Ports A und B verfügen außerdem noch über ein Locking-Register als auch über zwei Alternate-function-Register.
Über die Konfigurationsregister gpiox_otyper, gpiox_pupdr und gpiox_ospeeder werden der Typ, Pull-up/down und die Geschwindigkeit des entsprechenden Pins ausgewählt. Mithilfe von gpiox_moder wird der Pin entweder als Eingang, Ausgang, alternative Funktion (z. B. USART) oder analog (z. B. für ADC) konfiguriert. Die Datenregister gpio_idr und gpio_odr enthalten die Ein- und Ausgangswerte des jeweiligen Pins. Das set/reset-Register wird zum bitweisen Setzen, bzw. Rücksetzen der aussgangsseitigen Bits des Bits, was für diese Aufgabe nicht benötigt wird. Gleiches trifft auch auf das Locking-Register, welches den Schreibzugriff auf einen Pin verhindern kann, zu. Die gpio_afrl- und gpio_afrh-Register dienen der Auswahl einer alternativen Funktion am entsprechenden Port.

2.4.2 Konfiguration der Register

Anmerkung: Fehlende Register wurden in ihrer Default-Einstellung belassen.

GPIO port mode register

Auszug aus RM S. 136

8.4.1 GPIO port mode register (GPIOx_MODER) (x =A..D, F)

Address offset:0x00

Reset values:

- 0x2800 0000 for port A
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
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
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y+1:2y **MODERy[1:0]:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O mode.
00: Input mode (reset state)
01: General purpose output mode
10: Alternate function mode
11: Analog mode

Auszug aus STM-Datenblatt S. 29 und 31:

Table 11. STM32F030x4/6/8/C pin definitions (continued)

Pin number				Pin name (function after reset)	Pin type	I/O structure	Notes	Pin functions	
LQFP64	LQFP48	LQFP32	TSSOP20					Alternate functions	Additional functions
8	-	-	-	PC0	I/O	TTa	-	EVENTOUT, USART6_TX(5)	ADC_IN10
9	-	-	-	PC1	I/O	TTa	-	EVENTOUT, USART6_RX(5)	ADC_IN11
10	-	-	-	PC2	I/O	TTa	-	SPI2_MISO(5), EVENTOUT	ADC_IN12
11	-	-	-	PC3	I/O	TTa	-	SPI2_MOSI(5), EVENTOUT	ADC_IN13
12	8	-	-	VSSA	S	-	-	Analog ground	
13	9	5	5	VDDA	S	-	-	Analog power supply	
14	10	6	6	PA0	I/O	TTa	-	USART1_CTS(2), USART2_CTS(3)(5), USART4_TX(5)	ADC_IN0, RTC_TAMP2, WKUP1
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)	-

Zur Erfüllung der Aufgabenstellung müssen die Ports PA9, PA10 und PA12 auf alternate function sowie der Port PA0 auf analog eingestellt werden. PA9 wird zum Senden (TX) und PA10 zum Empfangen (RX) von Daten über USART verwendet, PA12 dient als Driver-Enable (DE) des angeschlossenen MAX485. PA0 wird als Eingang des ADC verwendet und wird daher als analog konfiguriert.

GPIO port output speed register

Auszug aus RM S. 137

8.4.3 GPIO port output speed register (GPIOx_OSPEEDR) (x = A..D, F)

Address offset: 0x08

Reset value:

- 0x0C00 0000 for port A
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OSPEEDR15 [1:0]		OSPEEDR14 [1:0]		OSPEEDR13 [1:0]		OSPEEDR12 [1:0]		OSPEEDR11 [1:0]		OSPEEDR10 [1:0]		OSPEEDR9 [1:0]		OSPEEDR8 [1:0]	
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
OSPEEDR7 [1:0]		OSPEEDR6 [1:0]		OSPEEDR5 [1:0]		OSPEEDR4 [1:0]		OSPEEDR3 [1:0]		OSPEEDR2 [1:0]		OSPEEDR1 [1:0]		OSPEEDR0 [1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y+1:2y **OSPEEDRy[1:0]**: Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O output speed.
x0: Low speed
01: Medium speed
11: High speed
Note: Refer to the device datasheet for the frequency specifications and the power supply and load conditions for each speed.

Anmerkung: In diesem Projekt werden PA9 und PA12 auf High Speed eingestellt.

GPIO alternate function high register

Auszug aus RM S. 141

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

Auszug aus STM-Datenblatt S. 34:

Pin name	AF0	AF1	AF2	AF3	AF4	AF5	AF6
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)	-	-
PA12	EVENTOUT	USART1_RTS	TIM1_ETR	-	-	SDA	-

Wie aus dem Datenblatt-Auszug zu erkennen, muss die AF1 auf den Ports 9, 10 und 12 ausgewählt werden, um USART1 auf diesen zu aktivieren (Die ADC-Funktion auf PA0 wird als "additional function" bezeichnet, diese muss nicht durch das AFLR-Register aktiviert werden).

2.4.3 Implementierung in C

```
// -- alternate function configuration --
// Port A boundary start address: 0x48000000

uint32_t *gpioA_afrh; // GPIO alternate function high
    register
gpioA_afrh = 0x48000000 + 0x24;
```

```
*gpioA_afrh |= 0x00010110; // Auswahl der AF1 (USART1) auf
    PA9 (TX), PA10 (RX) und PA12 (DE)

uint32_t *gpioA_ospeedr; // GPIO port output speed register
gpioA_ospeedr = 0x48000000 + 0x08;
*gpioA_ospeedr |= 0x030C0000; // Auswahl High Speed auf PA9
     (TX) und PA12 (DE)

uint32_t *gpioA_moder; // GPIO port mode register
gpioA_moder = 0x48000000 + 0x00;
*gpioA_moder |= 0x2A280003; // Enable der alternate
    functions USART1 auf PA9 (TX), PA10 (RX) und PA12(DE)
    sowie Aktivierung analog function auf PA0 fuer den ADC
```

2.5 Takt-Konfiguration

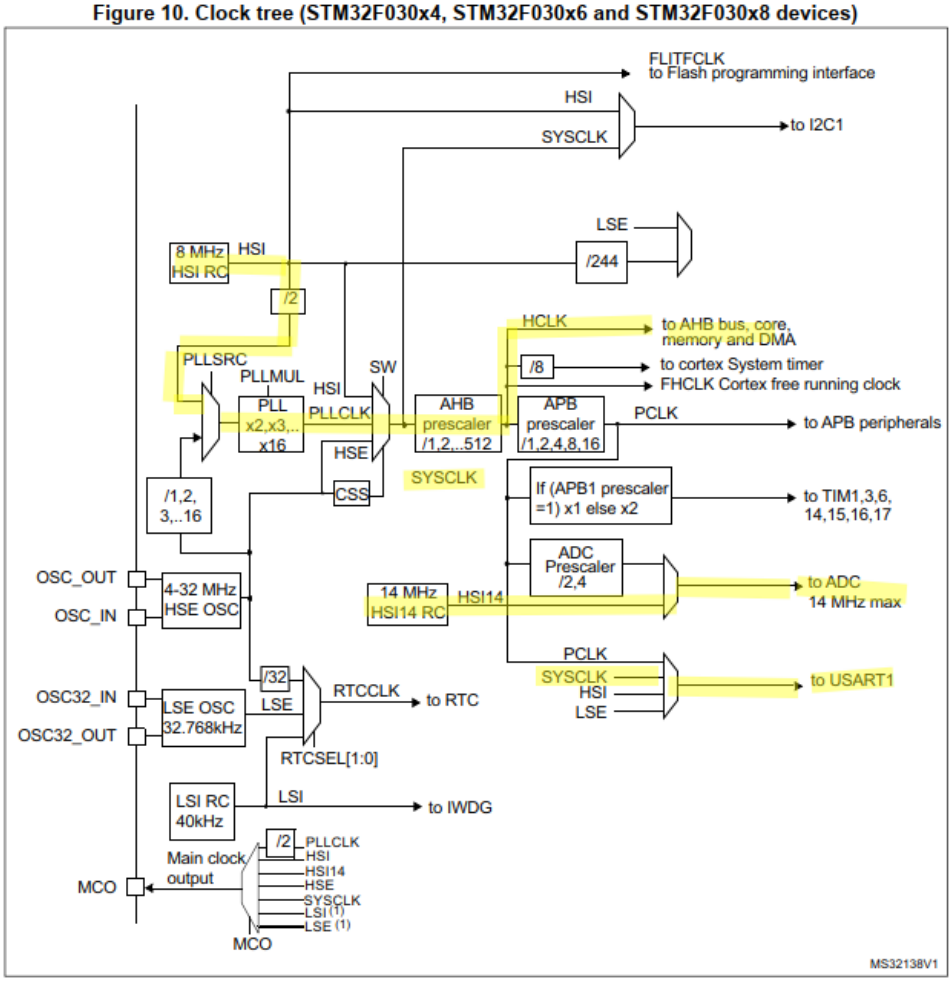
siehe RM S. 89ff.

2.5.1 Allgemeines

Der Systemtakt kann entweder durch eine interne 8MHz-Taktquelle (HSI-Clock), einer externen Taktquelle (HSE-Clock), oder einer PLL-Clock gespeist werden. Weiters verfügt der ADC über eine eigene 14MHz-Taktquelle. Die Takte der Busse AHB und APB können durch Prescaler angepasst werden. Normalerweise wird Peripherie mit der gleichen Frequenz des entsprechenden Busses getaktet, dies gilt aber nicht für USART1 und den ADC, welche durch verschiedene Taktquellen gespeist werden können.

Auszug aus RM S. 90:

Figure 10. Clock tree (STM32F030x4, STM32F030x6 and STM32F030x8 devices)



Der Clock Tree stellt die Konfigurationsmöglichkeiten der Clocks graphisch dar, die gelben Markierungen dienen zur Kennzeichnung der für dieses Projekt vorgenommenen Einstellungen. Wie am Clock Tree zu erkennen, kann die PLL entweder von der HSE-Clock oder von der HSI-Clock gespeist werden. Die System Clock wird entweder von der HSE-Clock, der HSI-Clock oder der PLL gespeist und bestimmt in weiterer Folge die Maximalfrequenz der Busse AHB und APB. Der Takt für diese kann durch Prescaler weiter herunter geteilt werden, wobei die Frequenz des APB-Busses maximal jener des AHB-Busses entsprechen kann. Der Takt des ADC wird entweder vom APB-Bus gespeist oder von seiner eigenen 14MHz-Taktquelle. Die USART1 kann wahlweise entweder vom APB-Bus, von der System Clock, von der

HSI-Clock oder von der HSE-Clock gespeist werden.
Für dieses Projekt wurde entschieden, die interne HSI-Clock anstatt einer externen zu verwenden, um externe Beschaltung einzusparen. Als System Clock wird die PLL verwendet, welche den durch die HSI-Clock gespeiste Taktfrequenz auf 48MHz erhöht. Diese wird dann anschließend als Takt für die USART1 verwendet. Der ADC wird durch seine 14MHz-Taktquelle gespeist.

Modifikation der PLL-Konfiguration

Auszug aus RM S. 94:

To modify the PLL configuration, proceed as follows:

1. Disable the PLL by setting PLLON to 0.
2. Wait until PLLRDY is cleared. The PLL is now fully stopped.
3. Change the desired parameter.
4. Enable the PLL again by setting PLLON to 1.
5. Wait until PLLRDY is set.

2.5.2 Konfiguration der Register

Anmerkung: fehlende Register wurden in ihrer Default-Einstellung belassen

Clock Control Register

Auszug aus RM S. 99/100

7.4.1 Clock control register (RCC_CR)

Address offset: 0x00

Reset value: 0x0000 XX83 where X is undefined.

Access: no wait state, word, half-word and byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	PLL RDY	PLLON	Res.	Res.	Res.	Res.	CSS ON	HSE BYP	HSE RDY	HSE ON
						r	rw					rw	rw	r	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HSICAL[7:0]								HSITRIM[4:0]					Res.	HSI RDY	HSION
r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw		r	rw

Bit 25 **PLLRDY:** PLL clock ready flag
Set by hardware to indicate that the PLL is locked.
0: PLL unlocked
1: PLL locked

Bit 24 **PLLON:** PLL enable
Set and cleared by software to enable PLL.
Cleared by hardware when entering Stop or Standby mode. This bit can not be reset if the PLL clock is used as system clock or is selected to become the system clock.
0: PLL OFF
1: PLL ON

Bit 1 **HSIRDY:** HSI clock ready flag
Set by hardware to indicate that HSI oscillator is stable. After the HSION bit is cleared, HSIRDY goes low after 6 HSI oscillator clock cycles.
0: HSI oscillator not ready
1: HSI oscillator ready

Bit 0 **HSION:** HSI clock enable
Set and cleared by software.
Set by hardware to force the HSI oscillator ON when leaving Stop or Standby mode or in case of failure of the HSE crystal oscillator used directly or indirectly as system clock. This bit cannot be reset if the HSI is used directly or indirectly as system clock or is selected to become the system clock.
0: HSI oscillator OFF
1: HSI oscillator ON

Mittels PLLRDY wird der Status der PLL überprüft, PLLON wird verwendet, um die PLL zu enabeln. Gleiches gilt für HSIRDY und HSION im Bezug auf die HSI-Clock. Die anderen Bits haben in diesem Projekt keine Bedeutung.

Clock Configuration Register

Auszug aus RM S. 101-103

7.4.2 Clock configuration register (RCC_CFGR)

Address offset: 0x04

Reset value: 0x0000 0000

Access: 0 ≤ wait state ≤ 2, word, half-word and byte access

1 or 2 wait states inserted only if the access occurs during clock source switch.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PLL NODIV	MCOPRE[2:0]			MCO[3:0]				Res.	Res.	PLLMUL[3:0]				PLL XTPRE	PLL SRC
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
Res.	ADC PRE	Res.	Res.	Res.	PPRE[2:0]			HPRE[3:0]				SWS[1:0]		SW[1:0]	
	rw				rw	rw	rw	rw	rw	rw	rw	r	r	rw	rw

Bits 21:18 **PLLMUL[3:0]:** PLL multiplication factor

These bits are written by software to define the PLL multiplication factor. These bits can be written only when PLL is disabled.

Caution: The PLL output frequency must not exceed 48 MHz.

0000: PLL input clock x 2
0001: PLL input clock x 3
0010: PLL input clock x 4
0011: PLL input clock x 5
0100: PLL input clock x 6
0101: PLL input clock x 7
0110: PLL input clock x 8
0111: PLL input clock x 9
1000: PLL input clock x 10
1001: PLL input clock x 11
1010: PLL input clock x 12
1011: PLL input clock x 13
1100: PLL input clock x 14
1101: PLL input clock x 15
1110: PLL input clock x 16
1111: PLL input clock x 16

Bit 16 **PLLSRC:** PLL entry clock source

Set and cleared by software to select PLL clock source. This bit can be written only when PLL is disabled.

0: HSI/2 selected as PLL input clock
1: HSE/PREDIV selected as PLL input clock (refer to *Section 7.4.12: Clock configuration register 2 (RCC_CFGR2) on page 122*)

Bits 10:8 **PPRE[2:0]:** PCLK prescaler

Set and cleared by software to control the division factor of the APB clock (PCLK).

0xx: HCLK not divided
100: HCLK divided by 2
101: HCLK divided by 4
110: HCLK divided by 8
111: HCLK divided by 16

Bits 7:4 **HPRE[3:0]:** HCLK prescaler
Set and cleared by software to control the division factor of the AHB clock.
0xxx: SYSCLK not divided
1000: SYSCLK divided by 2
1001: SYSCLK divided by 4
1010: SYSCLK divided by 8
1011: SYSCLK divided by 16
1100: SYSCLK divided by 64
1101: SYSCLK divided by 128
1110: SYSCLK divided by 256
1111: SYSCLK divided by 512

Bits 3:2 **SWS[1:0]:** System clock switch status
Set and cleared by hardware to indicate which clock source is used as system clock.
00: HSI oscillator used as system clock
01: HSE oscillator used as system clock
10: PLL used as system clock
11: Reserved, must be kept at reset value.

Bits 1:0 **SW[1:0]:** System clock switch
Set and cleared by software to select SYSCLK source.
Cleared by hardware to force HSI selection when leaving Stop and Standby mode or in case of failure of the HSE oscillator used directly or indirectly as system clock (if the Clock Security System is enabled).
00: HSI selected as system clock
01: HSE selected as system clock
10: PLL selected as system clock
11: Reserved, must be kept at reset value.

Um einen System-Takt von 48MHz zu erhalten, wird bei der PLL ein Multiplikationsfaktor von 12 benötigt, weil diese mit der Hälfte des Taktes der 8MHz-HSI-Clocks gespeist wird. Weiters werden die Takte der AHB- und APB-Busse nicht geteilt und entsprechen damit ebenfalls 48MHz.

APB Peripheral Clock Enable Register 2

Auszug aus RM S. 112-114

7.4.7 APB peripheral clock enable register 2 (RCC_APB2ENR)

Address: 0x18

Reset value: 0x0000 0000

Access: word, half-word and byte access

No wait states, except if the access occurs while an access to a peripheral in the APB domain is on going. In this case, wait states are inserted until the access to APB peripheral is finished.

Note: When the peripheral clock is not active, the peripheral register values may not be readable by software and the returned value is always 0x0.

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.	DBG MCUEN	Res.	Res.	Res.	TIM17 EN	TIM16 EN	TIM15EN
									rw				rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	USART1 EN	Res.	SPI1EN	TIM1EN	Res.	ADCEN	Res.	Res.	Res.	USART6 EN	Res.	Res.	Res.	Res.	SYSCFG COMPEN
	rw		rw	rw		rw				rw					rw

Bit 14 **USART1EN:** USART1 clock enable
Set and cleared by software.
0: USART1clock disabled
1: USART1clock enabled

Bit 13 Reserved, must be kept at reset value.

Bit 9 **ADCEN:** ADC interface clock enable
Set and cleared by software.
0: ADC interface disabled
1: ADC interface clock enabled

Clock Configuration Register 3

Auszug aus RM S. 123

7.4.13 Clock configuration register 3 (RCC_CFGR3)

Address: 0x30

Reset value: 0x0000 0000

Access: no wait states, word, half-word and byte access

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
Res.	Res.	Res.	Res.	Res.	Res.	Res.	ADC SW	USB SW	Res.	Res.	I2C1 SW	Res.	Res.	USART1SW[1:0]	
							rw	rw			rw			rw	rw

Bit 8 **ADCSW:** ADC clock source selection

Obsolete setting. To be kept at reset value, connecting the HSI14 clock to the ADC asynchronous clock input. Proper ADC clock selection is done inside the ADC_CFGR2 (refer to *Section 12.11.5: ADC configuration register 2 (ADC_CFGR2) on page 216*).

Bits 1:0 **USART1SW[1:0]:** USART1 clock source selection
This bit is set and cleared by software to select the USART1 clock source.
00: PCLK selected as USART1 clock source (default)
01: System clock (SYSCLK) selected as USART1 clock
10: LSE clock selected as USART1 clock
11: HSI clock selected as USART1 clock

Anmerkung: Die USART1 wird in diesem Projekt von der System clock gespeist.

Clock Control Register 2

Auszug aus RM S. 123/124

7.4.14 Clock control register 2 (RCC_CR2)

Address: 0x34

Reset value: 0xXX00 XX80, where X is undefined.

Access: no wait states, word, half-word and byte access

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
HSI14CAL[7:0]								HSI14TRIM[4:0]					HSI14 DIS	HSI14 RDY	HSI14 ON
r	r	r	r	r	r	r	r	rw	rw	rw	rw	rw	rw	r	rw

Bit 1 **HSI14RDY:** HSI14 clock ready flag
Set by hardware to indicate that HSI14 oscillator is stable. After the HSI14ON bit is cleared, HSI14RDY goes low after 6 HSI14 oscillator clock cycles.
0: HSI14 oscillator not ready
1: HSI14 oscillator ready

Bit 0 **HSI14ON:** HSI14 clock enable
Set and cleared by software.
0: HSI14 oscillator OFF
1: HSI14 oscillator ON

Anmerkung: In diesem Projekt wird die HSI14-Clock als Taktquelle für den ADC verwendet.

ADC Configuration Register 2

Auszug aus RM S. 216

12.11.5 ADC configuration register 2 (ADC_CFGR2)

Address offset: 0x10

Reset value: 0x0000 0000

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

Bits 31:30 **CKMODE[1:0]:** ADC clock mode

These bits are set and cleared by software to define how the analog ADC is clocked:
00: ADCCLK (Asynchronous clock mode), generated at product level (refer to RCC section)
01: PCLK/2 (Synchronous clock mode)
10: PCLK/4 (Synchronous clock mode)
11: Reserved
In all synchronous clock modes, there is no jitter in the delay from a timer trigger to the start of a conversion.

Note: Software is allowed to write these bits only when the ADC is disabled (ADCAL=0, ADSTART=0, ADSTP=0, ADDIS=0 and ADEN=0)

Anmerkung: In diesem Projekt wird die Default-Einstellung, also der async. Clock Mode, verwendet.

2.5.3 Implementierung in C

```
// -- clock configuration --
// RCC boundary start address: 0x40021000

uint32_t *rcc_cfgr; // Clock configuration register
rcc_cfgr = 0x40021000 + 0x04;
*rcc_cfgr |= 0x00280000; // Festlegung des
    Multiplikationsfaktors der PLL (12 => f=12*8MHz/2=48MHz)

uint32_t *rcc_cr; // Clock control register
rcc_cr = 0x40021000 + 0x00;
*rcc_cr |= 0x01000000; // Enable PLL
while((*rcc_cr) & 0x02000000) != 0x02000000){ // Warten auf
     PLLRDY = 1
         }

*rcc_cfgr |= 0x00000002; // Festlegung der PLL als SYSCLK

uint32_t *rcc_cfgr3; // Clock configuration register 3
rcc_cfgr3 = 0x40021000 + 0x30;
```

```
*rcc_cfgr3 |= 0x00000001; // SYSCLK (=PLL) als USART1-CLK
    ausgewaehlt

uint32_t *rcc_cr2; // Clock control register 2
rcc_cr2 = 0x40021000 + 0x34;
*rcc_cr2 |= 0x00000001; // Enable HSI14

// Konfiguration ADC-Clock
// Verwendung der Default-Einstellung (async. clock mode ->
     HSI14-Clock)

uint32_t *rcc_apb2enr; // APB peripheral clock enable
    register 2
rcc_apb2enr = 0x40021000 + 0x18;
*rcc_apb2enr |= 0x00004200 // Enable CLK der USART1 und des
     ADC
```

2.6 Konfiguration des ADC

siehe RM S. 181ff

2.6.1 Allgemeines

Der ADC des STM32F030F4 verfügt über eine einstellbare Auflösung und kann über multiplexed Channels mehrere Signale messen. Die Umwandlung kann sowohl fortlaufend als auch einzeln erfolgen und das Wandelergebnis kann sowohl left- als auch right-aligned gespeichert werden.
Da ein 10-Byte großer Buffer übertragen werden soll, wird eine Auflösung von 8 Bit verwendet. Weiters wird nur ein Channel - Channel 0 - verwendet, die Umwandlung geschieht fortlaufend. Die Daten werden right-aligned gespeichert.

Kalibration

Auszug aus RM S. 185

Calibration software procedure

1. Ensure that ADEN=0 and DMAEN=0
2. Set ADCAL=1
3. Wait until ADCAL=0
4. The calibration factor can be read from bits 6:0 of ADC_DR.

ADC-Enable

Auszug aus RM S. 185

> Follow this procedure to enable the ADC:
>
> 1. Clear the ADRDY bit in ADC_ISR register by programming this bit to 1.
> 2. Set ADEN=1 in the ADC_CR register.
> 3. Wait until ADRDY=1 in the ADC_ISR register and continue to write ADEN=1 (ADRDY is set after the ADC startup time). This can be handled by interrupt if the interrupt is enabled by setting the ADRDYIE bit in the ADC_IER register.

ADC-Disable

Auszug aus RM S. 186

> Follow this procedure to disable the ADC:
>
> 1. Check that ADSTART=0 in the ADC_CR register to ensure that no conversion is ongoing. If required, stop any ongoing conversion by writing 1 to the ADSTP bit in the ADC_CR register and waiting until this bit is read at 0.
> 2. Set ADDIS=1 in the ADC_CR register.
> 3. If required by the application, wait until ADEN=0 in the ADC_CR register, indicating that the ADC is fully disabled (ADDIS is automatically reset once ADEN=0).
> 4. Clear the ADRDY bit in ADC_ISR register by programming this bit to 1 (optional).

ADC-Kofigurationsregeln

Auszug aus RM S. 188

> Software must write to the ADCAL and ADEN bits in the ADC_CR register if the ADC is disabled (ADEN must be 0).
>
> Software must only write to the ADSTART and ADDIS bits in the ADC_CR register only if the ADC is enabled and there is no pending request to disable the ADC (ADEN = 1 and ADDIS = 0).
>
> For all the other control bits in the ADC_IER, ADC_CFGRi, ADC_SMPR, ADC_TR, ADC_CHSELR and ADC_CCR registers, software must only write to the configuration control bits if the ADC is enabled (ADEN = 1) and if there is no conversion ongoing (ADSTART = 0).
>
> Software must only write to the ADSTP bit in the ADC_CR register if the ADC is enabled (and possibly converting) and there is no pending request to disable the ADC (ADSTART = 1 and ADDIS = 0)

Anmerkung: Bei Verletzung dieser Regeln kann es geschehen, dass der ADC in einen undefinierten Zustand eintritt. Sollte dies geschehen, muss der ADC disabelt werden und alle Bits im ADC_CR-Register zurückgesetzt werden.

Kanalauswahl

siehe RM S. 189

Diese wird mit dem CHSEL-Register durchgeführt. Sollten mehrere die Signale mehrere Kanäle umgewandelt werden, kann die Reihenfolge mit dem SCANDIR-Register eingestellt werden; weil in diesem Projekt nur Daten aus einem Kanal ausgwertet werden, ist dies bedeutungslos.

Programmierbare Sampling-Time

siehe RM S. 189

Diese wurde in diesem Projekt auf ihren Default-Wert belassen (1,5 ADC-Zyklen).

Single/Continuous Conversion Mode

siehe RM S. 190

Abhängig vom CONT-Bit im ADC_CFGR1-Register kann der Conversion Mode ausgewählt werden, in diesem Projekt wird der Continuous Mode verwendet. In diesem Fall startet der ADC nach Vorliegen des Wandelergebnisses den Wandelvorgang automatisch neu, das EOC-Flag wird gesetzt und es wird, sofern enabelt (dies ist in diesem Projekt der Fall), ein Interrupt ausgelöst.

Starten des Wandelvorgangs

siehe RM S. 190/191

Der Wandelvorgang wird durch das Setzen des ADSTART-Bits gestartet. Solange dieses gesetzt ist, ist der ADC aktiv.

Stoppen des Wandelvorgangs

siehe RM S. 192

Der Wandelvorgang wird durch das Setzen des ADSTP-Bits gestoppt, der aktuelle Wandelvorgang wird dadurch abgebrochen.

Programmieren der Auflösung

siehe RM S. 194

Diese kann durch das Setzen der RES-Bits konfiguriert werden, in diesem Projekt wird mit einer Auflösung von 8 Bits gearbeitet.

Ende des Wandelvorgangs

siehe RM S. 195

Nach Vorliegen des Wandelergebnisses wird das EOC(End of Conversion)-Flag gesetzt und, sofern enabelt, ein Interrupt ausgelöst. Das Flag wird durch das Auslesen des Inhalts des Datenregisters zurückgesetzt, dies kann aber auch manuell durch das Schreiben von '1' an das dieses erfolgen.
In diesem Projekt wird beim Ende des Wandelvorgangs ein Interrupt ausgelöst, das EOC-Flag wird manuell am Beginn der Interrupt-Service-Routine zurückgesetzt.

Data Alignment

siehe RM S . 198

Das Wandelergebnis wird im ADC_DR-Register entweder left- oder right-aligned gespeichert; in diesem Projekt wurde entschieden, die Daten right-aligned zu speichern.

Overrun

siehe RM S. 198-199
Sollte ein neues Wandelergebnis vorliegen, aber das vorherige noch nicht ausgelesen worden sein (EOC-Flag nicht zurückgesetzt), wird das weitere Verhalten des ADC - alte Daten überschreiben oder neue Daten verwerfen - durch den Overrun-Mode festgelegt.
In diesem Projekt wurde der Overrun-Mode so konfiguriert, dass neue Daten alte übeschreiben.

ADC Interrupts

siehe RM S. 206

Ein Interrupt kann bei mehreren Ereignissen erfolgen, es sollte aber beachtet werden, dass alle Ereignisse dieselbe Interrupt-Service-Routine auslösen. Weiters sind defaultmäßig alle Interrupts disabelt, dies müssen also vor ihrer Verwendung aktiviert werden.
In diesem Projekt werden Interrupts beim Vorliegen des Wandelergebnisses ausgelöst.

2.6.2 Konfiguration der Register

Anmerkung: Fehlende Register wurden in ihrer Default-Einstellung belassen.

ADC Interrupt And Status Register

Auszug aus RM S. 207

12.11.1 ADC interrupt and status register (ADC_ISR)

Address offset: 0x00

Reset value: 0x0000 0000

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
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	AWD	Res.	Res.	OVR	EOSEQ	EOC	EOSMP	ADRDY
								rc_w1			rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Bit 2 **EOC**: End of conversion flag

This bit is set by hardware at the end of each conversion of a channel when a new data result is available in the ADC_DR register. It is cleared by software writing 1 to it or by reading the ADC_DR register.

0: Channel conversion not complete (or the flag event was already acknowledged and cleared by software)

1: Channel conversion complete

Bit 0 **ADRDY**: ADC ready

This bit is set by hardware after the ADC has been enabled (bit ADEN=1) and when the ADC reaches a state where it is ready to accept conversion requests.

It is cleared by software writing 1 to it.

0: ADC not yet ready to start conversion (or the flag event was already acknowledged and cleared by software)

1: ADC is ready to start conversion

ADC Interrupt Enable Register

Auszug aus RM S. 208

12.11.2 ADC interrupt enable register (ADC_IER)

Address offset: 0x04

Reset value: 0x0000 0000

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
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	AWD IE	Res.	Res.	OVRIE	EOSEQ IE	EOCIE	EOSMP IE	ADRDY IE
								rw			rw	rw	rw	rw	rw

Bit 2 **EOCIE**: End of conversion interrupt enable
This bit is set and cleared by software to enable/disable the end of conversion interrupt.
0: EOC interrupt disabled
1: EOC interrupt enabled. An interrupt is generated when the EOC bit is set.
Note: Software is allowed to write this bit only when ADSTART=0 (which ensures that no conversion is ongoing).

Anmerkung: In diesem Projekt wird EOCIE enabelt.

ADC Control Register

Auszug aus RM S. 210

12.11.3 ADC control register (ADC_CR)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ADCAL	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
rs															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	ADSTP	Res.	ADSTART	ADDIS	ADEN
											rs		rs	rs	rs

Bit 31 **ADCAL**: ADC calibration

This bit is set by software to start the calibration of the ADC.

It is cleared by hardware after calibration is complete.

0: Calibration complete

1: Write 1 to calibrate the ADC. Read at 1 means that a calibration is in progress.

Note: Software is allowed to set ADCAL only when the ADC is disabled (ADCAL=0, ADSTART=0, ADSTP=0, ADDIS=0 and ADEN=0).

Bits 30:5 Reserved, must be kept at reset value.

Bit 4 **ADSTP**: ADC stop conversion command

This bit is set by software to stop and discard an ongoing conversion (ADSTP Command).

It is cleared by hardware when the conversion is effectively discarded and the ADC is ready to accept a new start conversion command.

0: No ADC stop conversion command ongoing

1: Write 1 to stop the ADC. Read 1 means that an ADSTP command is in progress.

Note: Setting ADSTP to '1' is only effective when ADSTART=1 and ADDIS=0 (ADC is enabled and may be converting and there is no pending request to disable the ADC)

Bit 3 Reserved, must be kept at reset value.

Bit 2 **ADSTART**: ADC start conversion command

This bit is set by software to start ADC conversion. Depending on the EXTEN [1:0] configuration bits, a conversion either starts immediately (software trigger configuration) or once a hardware trigger event occurs (hardware trigger configuration).

It is cleared by hardware:

– In single conversion mode (CONT=0, DISCEN=0), when software trigger is selected (EXTEN=00): at the assertion of the end of Conversion Sequence (EOSEQ) flag.

– In discontinuous conversion mode(CONT=0, DISCEN=1), when the software trigger is selected (EXTEN=00): at the assertion of the end of Conversion (EOC) flag.

– In all other cases: after the execution of the ADSTP command, at the same time as the ADSTP bit is cleared by hardware.

0: No ADC conversion is ongoing.

1: Write 1 to start the ADC. Read 1 means that the ADC is operating and may be converting.

Note: Software is allowed to set ADSTART only when ADEN=1 and ADDIS=0 (ADC is enabled and there is no pending request to disable the ADC)

Bit 1 **ADDIS**: ADC disable command

This bit is set by software to disable the ADC (ADDIS command) and put it into power-down state (OFF state).

It is cleared by hardware once the ADC is effectively disabled (ADEN is also cleared by hardware at this time).

0: No ADDIS command ongoing

1: Write 1 to disable the ADC. Read 1 means that an ADDIS command is in progress.

Note: Setting ADDIS to '1' is only effective when ADEN=1 and ADSTART=0 (which ensures that no conversion is ongoing)

Bit 0 **ADEN**: ADC enable command

This bit is set by software to enable the ADC. The ADC will be effectively ready to operate once the ADRDY flag has been set.

It is cleared by hardware when the ADC is disabled, after the execution of the ADDIS command.

0: ADC is disabled (OFF state)

1: Write 1 to enable the ADC.

Note: Software is allowed to set ADEN only when all bits of ADC_CR registers are 0 (ADCAL=0, ADSTP=0, ADSTART=0, ADDIS=0 and ADEN=0)

ADC Configuration Register 1

Auszug aus RM S. 212-215

12.11.4 ADC configuration register 1 (ADC_CFGR1)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	AWDCH[4:0]					Res.	Res.	AWDEN	AWDSGL	Res.	Res.	Res.	Res.	Res.	DISCEN
	rw	rw	rw	rw	rw			rw	rw						rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AUTOFF	WAIT	CONT	OVRMOD	EXTEN[1:0]		Res.	EXTSEL[2:0]			ALIGN	RES[1:0]		SCANDIR	DMACFG	DMAEN
rw	rw	rw	rw	rw			rw			rw	rw		rw	rw	rw

Bit 13 **CONT**: Single / continuous conversion mode

This bit is set and cleared by software. If it is set, conversion takes place continuously until it is cleared.

0: Single conversion mode

1: Continuous conversion mode

Note: It is not possible to have both discontinuous mode and continuous mode enabled: it is forbidden to set both bits DISCEN=1 and CONT=1.

Note: Software is allowed to write this bit only when ADSTART=0 (which ensures that no conversion is ongoing).

Bit 12 **OVRMOD**: Overrun management mode

This bit is set and cleared by software and configure the way data overruns are managed.

0: ADC_DR register is preserved with the old data when an overrun is detected.

1: ADC_DR register is overwritten with the last conversion result when an overrun is detected.

Note: Software is allowed to write this bit only when ADSTART=0 (which ensures that no conversion is ongoing).

Bit 5 **ALIGN**: Data alignment

This bit is set and cleared by software to select right or left alignment. Refer to *Figure 34: Data alignment and resolution on page 198*

0: Right alignment

1: Left alignment

Note: Software is allowed to write this bit only when ADSTART=0 (which ensures that no conversion is ongoing).

Bit 4:3 **RES[1:0]**: Data resolution

These bits are written by software to select the resolution of the conversion.

00: 12 bits

01: 10 bits

10: 8 bits

11: 6 bits

Note: Software is allowed to write these bits only when ADEN=0.

Anmerkung: In diesem Projekt werden CONT=1 (continuous mode), OVERMOD=1 (overwrite old data), ALIGN=0 (right-aligned) und RES=10 (8 Bit) gesetzt.

ADC Configuration Register 2

Auszug aus RM S. 216

12.11.5 ADC configuration register 2 (ADC_CFGR2)

Address offset: 0x10

Reset value: 0x0000 0000

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

Bits 31:30 **CKMODE[1:0]:** ADC clock mode

These bits are set and cleared by software to define how the analog ADC is clocked:
00: ADCCLK (Asynchronous clock mode), generated at product level (refer to RCC section)
01: PCLK/2 (Synchronous clock mode)
10: PCLK/4 (Synchronous clock mode)
11: Reserved
In all synchronous clock modes, there is no jitter in the delay from a timer trigger to the start of a conversion.

Note: Software is allowed to write these bits only when the ADC is disabled (ADCAL=0, ADSTART=0, ADSTP=0, ADDIS=0 and ADEN=0).

Bits 29:0 Reserved, must be kept at reset value.

Anmerkung: Die Takt-Konfiguration des ADC wurde bereits im Kapitel 2.5 erläutert.

ADC Channel Selection Register

Auszug aus RM S. 218

12.11.8 ADC channel selection register (ADC_CHSELR)

Address offset: 0x28

Reset value: 0x0000 0000

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.	CHSEL 17	CHSEL 16
														rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CHSEL 15	CHSEL 14	CHSEL 13	CHSEL 12	CHSEL 11	CHSEL 10	CHSEL 9	CHSEL 8	CHSEL 7	CHSEL 6	CHSEL 5	CHSEL 4	CHSEL 3	CHSEL 2	CHSEL 1	CHSEL 0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:18 Reserved, must be kept at reset value.

Bits 17:0 **CHSELx**: Channel-x selection

These bits are written by software and define which channels are part of the sequence of channels to be converted.

0: Input Channel-x is not selected for conversion

1: Input Channel-x is selected for conversion

Note: Software is allowed to write these bits only when ADSTART=0 (which ensures that no conversion is ongoing).

Anmerkung: In diesem Projekt wird nur Channel 0 verwendet.

ADC Data Register

Auszug aus RM S. 218

12.11.9 ADC data register (ADC_DR)

Address offset: 0x40

Reset value: 0x0000 0000

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
DATA[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

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

Bits 15:0 **DATA[15:0]**: Converted data

These bits are read-only. They contain the conversion result from the last converted channel. The data are left- or right-aligned as shown in *Figure 34: Data alignment and resolution on page 198.*

Just after a calibration is complete, DATA[6:0] contains the calibration factor.

Anmerkung: Daten werden right-aligned gespeichert.

2.6.3 Implementierung in C

```
// -- ADC configuration --
// ADC boundary start adress: 0x40012400

// Calibration software procedure
// Ensure that ADEN=0 and DMAEN=0 -> automatisch nach Reset
    der Fall
uint32_t *adc_cr; // ADC control register
adc_cr = 0x40012400 + 0x08;
*adc_cr |= 0x80000000; // Set ADCAL=1
while((*adc_cr) & 0x80000000 == 0x80000000){ // Wait until
   ADCAL=0
}

// Enable the ADC
uint32_t *adc_isr; // ADC interrupt and status register
adc_isr = 0x040012400 + 0x00;
*adc_isr |= 0x00000001; // Clear ADRDY bit
*adc_cr |= 0x00000001; // Set ADEN=1
while(*adc_isr == 0x00000001){// Wait until ADRDY=1
}

// miscellaneous
uint32_t *adc_cfgr1; // ADC configuration register 1
adc_cfgr1 = 0x40012400 + 0x0C;
*adc_cfgr1 |= 0x00003010; // continuous mode, overrun mode
   (overwrite), data alignment (right), resolution (8 bits)

// Sampling time selection
Defaulteinstellungen: sampling time = 1,5 clock cycles

// Channel Selection
uint32_t *adc_chselr; // ADC channel selection register
adc_chselr = 0x40012400 + 0x28;
*adc_chselr |= 0x00000001; // Channel 0 ausgewaehlt

// Enable Interrupts
uint32_t *adc_ier; // ADC interrupt enable register
adc_ier = 0x40012400 + 0x04;
*adc_ier |= 0x00000004; // Interrupt bei Vorliegen des
   Wandelergebnisses

        // Set ADSTART
        *adc_cr |= 0x00000004;
```

2.7 Konfiguration der USART

siehe RM S. 595ff

2.7.1 Allgemeines

functional description

siehe RM S. 598f

Jede bidirektionale Kommunikation über USART benötigt mindestens 2 Pins, einen Sende- (TX) und einen Empfangspin (RX). Sollte ersterer disabelt sein, so kehrt dieser in seine I/O-Konfiguration zurück; sollte dieser enabelt sein, aber keine Daten senden, so liegt dieser auf High.
Die Datenübertragung entspricht dem USART-Standard; die übertragenen Frames bestehen aus idle lines (vor Empfang oder Senden), einem Start-Bit, einem Datenwort (8 oder 9 Bit) und einem oder zwei Stop-Bits.
Weiters stehen zusätzliche Pins für RS232 und und ein Driver-Enable-Pin (DE) für RS485 zur Verfügung.
Für dieses Projekt werden folgende Einstellungen konfiguriert:

- Verwendete Pins: TX, RX, DE
- Datenwortlänge = 8 Bits + 1 Paritätsbit
- Anzahl Stopbits = 1
- Baudrate = 38,4k
- Parity = odd (ungerade)
- Interrupt bei Empfang von Daten

USART character description

siehe RM S. 599f

Über das M0-Bit wird die Länge des Datenwortes bestimmt; Bei M0=0 beträgt diese 8 Bit, durch das Aktivieren der Paritätskontrolle wird diese aber um 1 verringert (siehe 2.7.1), weshalb M0=1 eingestellt wird. Auf eine genauere Beschreibung wird mangels Projektrelevanz verzichtet.

USART Transmitter

siehe RM S. 600ff

Anmerkung: Bei USART werden die geringwertigsten Bits zuerst gesendet.

Auszug aus RM S. 602

Character transmission procedure

1. Program the M bit in USART_CR1 to define the word length.
2. Select the desired baud rate using the USART_BRR register.
3. Program the number of stop bits in USART_CR2.
4. Enable the USART by writing the UE bit in USART_CR1 register to 1.
5. Select DMA enable (DMAT) in USART_CR3 if multibuffer communication is to take place. Configure the DMA register as explained in multibuffer communication.
6. Set the TE bit in USART_CR1 to send an idle frame as first transmission.
7. Write the data to send in the USART_TDR register (this clears the TXE bit). Repeat this for each data to be transmitted in case of single buffer.
8. After writing the last data into the USART_TDR register, wait until TC=1. This indicates that the transmission of the last frame is complete. This is required for instance when the USART is disabled or enters the Halt mode to avoid corrupting the last transmission.

Anmerkung zum Senden mehrerer einzelner Bytes: Sobald das TXE-Bit gesetzt ist, können neue Daten in das TDR-Register transferiert werden, weil dieses die erfolgreiche Verschiebung der vorherigen Daten des TDR-Registers in das Shift-Register signalisiert.

USART Receiver

siehe RM S. 603ff

Nach dem Epfang der Startbitfolge (1110X0X0X0X0X0X0) wird das RXNE-Flag gesetzt und, sofern aktiviert, ein Interrupt ausgelöst. Da bei USART normalerweise die geringwertigsten Bits zuerst gesendet werden, werden diese auch als Erste empfangen.

Auszug aus RM S. 605

Character reception procedure

1. Program the M bit in USART_CR1 to define the word length.
2. Select the desired baud rate using the baud rate register USART_BRR
3. Program the number of stop bits in USART_CR2.
4. Enable the USART by writing the UE bit in USART_CR1 register to 1.
5. Select DMA enable (DMAR) in USART_CR3 if multibuffer communication is to take place. Configure the DMA register as explained in multibuffer communication.
6. Set the RE bit USART_CR1. This enables the receiver which begins searching for a start bit.

Das RXNE-Flag wird durch Hardware gesetzt und bedeutet, dass Daten empfangen wurden und diese gelesen werden können. Es wird entweder durch das Lesen der Daten oder durch das Schreiben von '1' an RXFRQ zurückgesetzt. Sollten Daten empfangen werden, aber das RXNE-Flag noch nicht zurückgesetzt worden sein, so würde ein Overrun-Error ausgelöst werden. Da in diesem Projekt die Kommunikation half-duplex geschieht und die Aufgabe des uC darin besteht, bei Empfang eines Zeichens Daten zu senden, dürfte dieser Fall bei korrekten Verhalten der Gegenseite nicht auftreten und spielt daher keine Rolle.

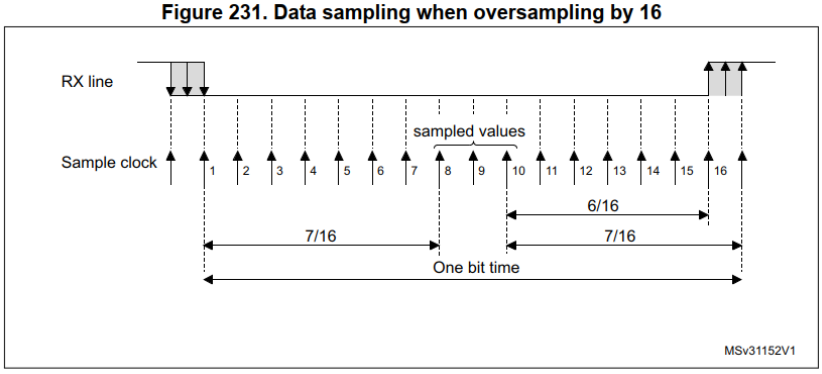
Auswahl der Taktquelle und Oversampling-Methode

siehe RM S. 606ff

Die Auswahl der Taktquelle muss vor dem Enable der USART erfolgen. Die Auswahl hängt von der gewünschten Übertragungsgeschwindigkeit (Baud-Rate) ab. Die Frequenz des Taktes wird sowohl im RM als auch in diesem Dokument als fck bezeichnet.
Die Oversampling-Methode wird durch das OVER8-Bit im Control 1 Register bestimmt und beträgt entweder 8 oder 16 Mal des Baud-Rate-Taktes (Für dieses Projekt wurde eine OSR von 16 gewählt).

Auszug aus RM S: 607

Figure 231. Data sampling when oversampling by 16



Das ONEBIT-Bit im Control 3 Register bestimmt die Methode zur Ermittlung des Logic-Levels. Diese unterscheiden sich durch die Anzahl an durchgeführten Proben in der Mitte des empfangenen Bits; entweder beträgt diese 3, wodurch Rauschen durch Vergleich dieser erkannt werden kann, oder diiese beträgt 1, wodurch die Toleranz für Abweichungen es Taktes erhöht wird. Für dieses Projekt wird Letzteres verwendet.

Framing Error

siehe RM S. 608

Wenn beim Empfang von Daten kein Stop-Bit empfangen wird (z. B. durch exessives Rauschen), so wird das FE-Bit gesetzt, welches durch das Schreiben von '1' an FECF im ICR-Register zurückgesetzt werden kann. Dies spielt für dieses Projekt aber keine Rolle, weil der Inhalt der empfangenen Daten nicht relevant ist.

Anzahl der Stop-Bits

siehe RM S. 609

Die Anzahl der Stop-Bits kann entweder 1 oder zwei betragen. In diesem Projekt wird diese als 1 konfiguriert, dies ist die Default-Einstellung.

Baud Rate-Konfiguration

siehe RM S. 609f

Die Baud-Rate ist sowohl für den Sende- als auch für den Empfangsvorgang gleich und wird im Baud Rate Register konfiguriert.

Auszug aus RM S. 609

Equation 1: Baud rate for standard USART (SPI mode included) (OVER8 = 0 or 1)

In case of oversampling by 16, the equation is:

$$\text{Tx/Rx baud} = \frac{f_{CK}}{\text{USARTDIV}}$$

In case of oversampling by 8, the equation is:

$$\text{Tx/Rx baud} = \frac{2 \times f_{CK}}{\text{USARTDIV}}$$

USARTDIV is an unsigned fixed point number that is coded on the USART_BRR register.

- When OVER8 = 0, BRR = USARTDIV.
- When OVER8 = 1
  - BRR[2:0] = USARTDIV[3:0] shifted 1 bit to the right.
  - BRR[3] must be kept cleared.
  - BRR[15:4] = USARTDIV[15:4]

Note: The baud counters are updated to the new value in the baud registers after a write operation to USART_BRR. Hence the baud rate register value should not be changed during communication.

In case of oversampling by 16 or 8, USARTDIV must be greater than or equal to 0d16.

fck ist die Taktfrequenz der USART, welche gleich der des Systemtakts ist, die 48MHz beträgt. Die Baudrate beträgt 38,4k, womit USARTDIV durch die Division von fck durch die Baudrate berechnet werden kann - dies ist aber nicht notwendig, weil auf der nächsten Seite des RM eine Tabelle enthalten ist, welche die Werte von UDIV für verschiedene Baudraten bei einer fck von 48MHz zeigt:

Table 87. Error calculation for programmed baud rates at f_{CK} = 48 MHz in both cases of oversampling by 16 or by 8(1)

Baud rate		Oversampling by 16 (OVER8 = 0)			Oversampling by 8 (OVER8 = 1)		
S.No	Desired	Actual	BRR	% Error = (Calculated - Desired)B.Rate / Desired B.Rate	Actual	BRR	% Error
5	38.4 KBps	38.4 KBps	0x4E2	0	38.4 KBps	0x9C2	0

Das Baud-Rate Register muss also mit einem Wert von 0x4E2 beschrieben werden.

Toleranz der Empfangseinheit bezüglich Abweichungen der Taktfrequenz

Auszug aus RM S. 611

Table 89. Tolerance of the USART receiver when BRR [3:0] is different from 0000

M bit	OVER8 bit = 0		OVER8 bit = 1	
	ONEBIT=0	ONEBIT=1	ONEBIT=0	ONEBIT=1
0	3.33%	3.88%	2%	3%
1	3.03%	3.53%	1.82%	2.73%

Die maximale Abweichung der Taktfrequenz darf 3,03% betragen.

Auto Baud Rate-Erkennung

siehe RM S. 612

Die USART wäre in der Lage, die Baud Rate automatisch zu erkennen, dies wird aber in diesm Projekt nicht verwendet.

Parität

siehe RM S. 615

Die Konfiguration der Parität erfolgt im Control 1 Register. Abhängig von der durch das M-Bit konfigurierte Frame-Länge sind verschiedene Formate möglich:

Table 90. Frame formats

M bit	PCE bit	USART frame(1)
0	0	| SB | 8-bit data | STB |
0	1	| SB | 7-bit data | PB | STB |
1	0	| SB | 9-bit data | STB |
1	1	| SB | 8-bit data PB | STB |

1. Legends: SB: start bit, STB: stop bit, PB: parity bit. In the data register, the PB is always taking the MSB position (8th or 7th, depending on the M bits value).

Daraus folgt, um 1 Byte Daten übertragen zu können, das M-Bit als '1' eingestellt werden sollte.
Bei Parität wird, wie aus der Tabelle ersichtlich, ein weiteres Bit dem Frame hinzugefügt wird; dessen Wert hängt von der Anzahl anderer, im Frame vorkommender, 1en ab: Bei Even (gerader) Parity wird diese auf eine ger-

ade Anzahl, bei Odd (ungerader) Parity wird diese auf eine ungerade Anzahl ergänzt.
Bei Empfang eines Frames mit einer falschen Anzahl 1en wird das PE-Flag im Interrupt And Status Register gesetzt und bedeutet, dass die Daten invalid sind; dies spielt für dieses Projekt keine Rolle, weil lediglich das Empfangen von Daten, unabhängig von deren Inhalt, von Belang ist.
Beim Senden von Daten wird das MSB des Frames (dies ist bei USART das letzte übertragene Bit) entsprechend der Parity geändert.

Für dieses Projekt wird Odd Parity konfiguriert.

Verwenden des RS485-Übertragungsstandards

siehe RM S. 623

> **RS485 Driver Enable**
>
> The driver enable feature is enabled by setting bit DEM in the USART_CR3 control register. This allows the user to activate the external transceiver control, through the DE (Driver Enable) signal. The assertion time is the time between the activation of the DE signal and the beginning of the START bit. It is programmed using the DEAT [4:0] bit fields in the USART_CR1 control register. The de-assertion time is the time between the end of the last stop bit, in a transmitted message, and the de-activation of the DE signal. It is programmed using the DEDT [4:0] bit fields in the USART_CR1 control register. The polarity of the DE signal can be configured using the DEP bit in the USART_CR3 control register.
>
> In USART, the DEAT and DEDT are expressed in sample time units (1/8 or 1/16 bit duration, depending on the oversampling rate).

Bei diesem Projekt wird die Default-Einstellung der Polarität verwendet, weiters beträgt die OSR=16; die Bitlänge ergibt sich aus dem Kehrwert der Baud-Rate, gebrochen durch die Anzahl an Bits pro Frame, welche 11 beträgt (1 Startbit, 8 Datenbits, 1 Parity Bit und 1 Stopbit) und beträgt somit (1/38,4k)/11=2,37us, eine Sampling Time Unit beträgt demnach 2,37us/16=147ns.
Die Enable-, bzw. Disable-Zeiten können dem Datenblatt des MAX485 auf S. 3 und 10 entnommen werden:

Driver Rise or Fall Time	t_R, t_F	Figures 6 and 8, R_{DIFF} = 54Ω, C_{L1} = C_{L2} = 100pF	MAX481, MAX485, MAX1487	3	15	40	ns
			MAX490C/E, MAX491C/E	5	15	25	
			MAX490M, MAX491M	3	15	40	
Driver Enable to Output High	t_{ZH}	Figures 7 and 9, C_L = 100pF, S2 closed			40	70	ns
Driver Enable to Output Low	t_{ZL}	Figures 7 and 9, C_L = 100pF, S1 closed			40	70	ns
Driver Disable Time from Low	t_{LZ}	Figures 7 and 9, C_L = 15pF, S1 closed			40	70	ns
Driver Disable Time from High	t_{HZ}	Figures 7 and 9, C_L = 15pF, S2 closed			40	70	ns

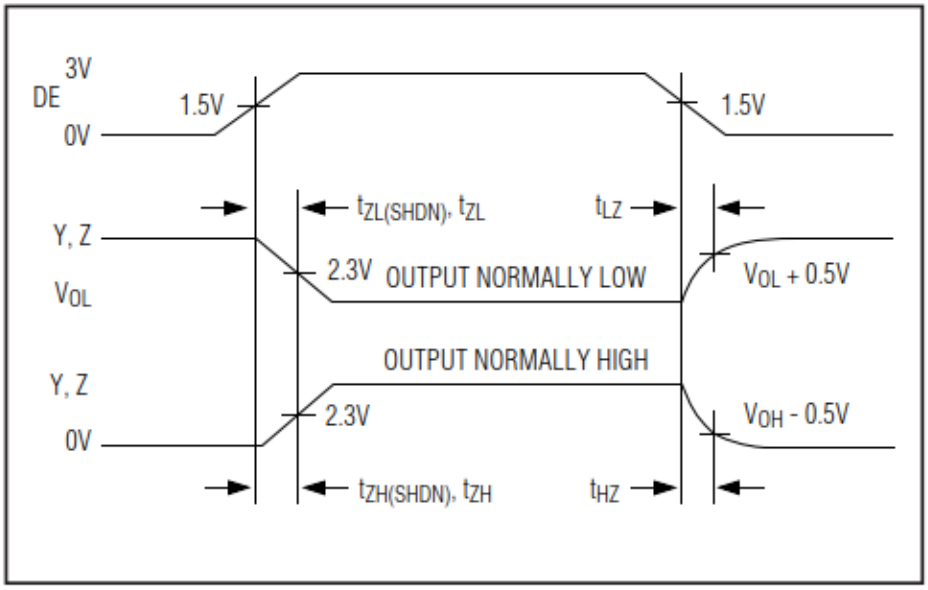


Figure 9. Driver Enable and Disable Times (except MAX488 and MAX490)

Da die Maximaldauer 70ns beträgt, wird lediglich eine Sampling Time Unit benötigt, daher werden DEAT als auch DEDT mit dem binären Wert "00001" beschrieben.

Interrupts

siehe RM S. 624

Alle Interrupt-Ereignisse lösen diesselbe Interrupt-Service-Routine aus; diese müssen durch das Setzen des entsprechenden Bits erst enabelt werden.
In diesem Projekt soll der Empfang von Daten Interrupts auslösen, dies wird durch das Enabeln des RXNE-Flags durch das Setzen des RXNEIE-Bits erreicht.

Auszug aus RM S. 624

Table 92. USART interrupt requests

Interrupt event	Event flag	Enable Control bit

Receive data register not empty (data ready to be read)	RXNE	RXNEIE
Overrun error detected	ORE	

2.7.2 Konfiguration der Register

Anmerkung: Fehlende Register wurden in ihrer Default-Einstellung belassen.

Control Register 1

Auszug aus RM S. 625-627

23.7.1 Control register 1 (USART_CR1)

Address offset: 0x00

Reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	M1	Res.	RTOIE	DEAT[4:0]					DEDT[4:0]				
			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
OVER8	CMIE	MME	M0	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	Res.	UE
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw

Bits 31:29 Reserved, must be kept at reset value

Bit 28 **M1**: Word length
This bit, with bit 12 (M0), determines the word length. It is set or cleared by software.
M[1:0] = 00: 1 Start bit, 8 data bits, n stop bits
M[1:0] = 01: 1 Start bit, 9 data bits, n stop bits
M[1:0] = 10: 1 Start bit, 7 data bits, n stop bits
This bit can only be written when the USART is disabled (UE=0).

Bits 25:21 **DEAT[4:0]**: Driver Enable assertion time
This 5-bit value defines the time between the activation of the DE (Driver Enable) signal and the beginning of the start bit. It is expressed in sample time units (1/8 or 1/16 bit duration, depending on the oversampling rate).
This bit field can only be written when the USART is disabled (UE=0).
Note: If the Driver Enable feature is not supported, this bit is reserved and must be kept cleared. Please refer to Section 23.3: USART implementation on page 597.

Bits 20:16 **DEDT[4:0]**: Driver Enable de-assertion time
This 5-bit value defines the time between the end of the last stop bit, in a transmitted message, and the de-activation of the DE (Driver Enable) signal. It is expressed in sample time units (1/8 or 1/16 bit duration, depending on the oversampling rate).
If the USART_TDR register is written during the DEDT time, the new data is transmitted only when the DEDT and DEAT times have both elapsed.
This bit field can only be written when the USART is disabled (UE=0).
Note: If the Driver Enable feature is not supported, this bit is reserved and must be kept cleared. Please refer to Section 23.3: USART implementation on page 597.

Bit 15 **OVER8**: Oversampling mode

0: Oversampling by 16
1: Oversampling by 8
This bit can only be written when the USART is disabled (UE=0).

Bit 12 **M0**: Word length

This bit determines the word length. It is set or cleared by software.
0: 1 Start bit, 8 data bits, n stop bits
1: 1 Start bit, 9 data bits, n stop bits

Bit 10 **PCE**: Parity control enable

This bit selects the hardware parity control (generation and detection). When the parity control is enabled, the computed parity is inserted at the MSB position (9th bit if M=1; 8th bit if M=0) and parity is checked on the received data. This bit is set and cleared by software. Once it is set, PCE is active after the current byte (in reception and in transmission).
0: Parity control disabled
1: Parity control enabled
This bit field can only be written when the USART is disabled (UE=0).

Bit 9 **PS**: Parity selection

This bit selects the odd or even parity when the parity generation/detection is enabled (PCE bit set). It is set and cleared by software. The parity will be selected after the current byte.
0: Even parity
1: Odd parity
This bit field can only be written when the USART is disabled (UE=0).

Bit 5 **RXNEIE**: RXNE interrupt enable

This bit is set and cleared by software.
0: Interrupt is inhibited
1: A USART interrupt is generated whenever ORE=1 or RXNE=1 in the USART_ISR register

Bit 3 **TE**: Transmitter enable

This bit enables the transmitter. It is set and cleared by software.

0: Transmitter is disabled

1: Transmitter is enabled

During transmission, a "0" pulse on the TE bit ("0" followed by "1") sends a preamble (idle line) after the current word. In order to generate an idle character, the TE must not be immediately written to 1.

Bit 2 **RE**: Receiver enable

This bit enables the receiver. It is set and cleared by software.

0: Receiver is disabled

1: Receiver is enabled and begins searching for a start bit

Bit 1 Reserved, must be kept at reset value.

Bit 0 **UE**: USART enable

When this bit is cleared, the USART prescalers and outputs are stopped immediately, and current operations are discarded. The configuration of the USART is kept, but all the status flags, in the USART_ISR are set to their default values. This bit is set and cleared by software.

0: USART prescaler and outputs disabled, low-power mode

1: USART enabled

Note: In order to go into low-power mode without generating errors on the line, the TE bit must be reset before and the software must wait for the TC bit in the USART_ISR to be set before resetting the UE bit.

The DMA requests are also reset when UE = 0 so the DMA channel must be disabled before resetting the UE bit.

Anmerkung: In diesem Projekt werden M[1:0]=01 (1 Startbit, 8 Datenbits, 1 Paritätsbit, 1 Stopbit), DEAT = 00001 (1 Sampling Time Unit), DEDT = 00001 (1 Sampling Time Unit), OVER8=0 (OSR=16), PCE=1 (Parity Control enabelt), PS=1 (Odd Parity ausgewählt) und RXNEIE=1 (Interrupt bei ORE=1 oder RXNE=1, also bei Empfang von Daten); TE, RE und UE werden je nach Bedarf gesetzt und zurückgesetzt.

Control Register 2

Auszug aus RM S. 628-630

23.7.2 Control register 2 (USART_CR2)

Address offset: 0x04

Reset value: 0x0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ADD[7:4]				ADD[3:0]				RTOEN	ABRMOD[1:0]		ABREN	MSBFIRST	DATAINV	TXINV	RXINV
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
SWAP	Res.	STOP[1:0]		CLKEN	CPOL	CPHA	LBCL	Res.	Res.	Res.	ADDM7	Res.	Res.	Res.	Res.
rw		rw	rw	rw	rw	rw	rw				rw				

Bits 13:12 **STOP[1:0]**: STOP bits
These bits are used for programming the stop bits.
00: 1 stop bit
01: Reserved
10: 2 stop bits
11: Reserved
This bit field can only be written when the USART is disabled (UE=0).

Anmerkung: In diesem Projekt wird die Anzahl der Stop-Bits auf 1 gesetzt, d. h., dass die Default-Konfiguration des Registers übernommen wird.

Control Register 3

Auszug aus RM S. 630-632

23.7.3 Control register 3 (USART_CR3)

Address offset: 0x08

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.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DEP	DEM	DDRE	OVR DIS	ONE BIT	CTSIE	CTSE	RTSE	DMAT	DMAR	Res.	Res.	HDSEL	Res.	Res.	EIE
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw			rw			rw

Bit 14 **DEM**: Driver enable mode
This bit allows the user to activate the external transceiver control, through the DE signal.
0: DE function is disabled.
1: DE function is enabled. The DE signal is output on the RTS pin.
This bit can only be written when the USART is disabled (UE=0).
Note: If the Driver Enable feature is not supported, this bit is reserved and must be kept cleared. Section 23.3: USART implementation on page 597.

Bit 12 **OVRDIS**: Overrun Disable
This bit is used to disable the receive overrun detection.
0: Overrun Error Flag, ORE, is set when received data is not read before receiving new data.
1: Overrun functionality is disabled. If new data is received while the RXNE flag is still set the ORE flag is not set and the new received data overwrites the previous content of the USART_RDR register.
This bit can only be written when the USART is disabled (UE=0).
Note: This control bit allows checking the communication flow without reading the data.

Bit 11 **ONEBIT**: One sample bit method enable

This bit allows the user to select the sample method. When the one sample bit method is selected the noise detection flag (NF) is disabled.

0: Three sample bit method

1: One sample bit method

This bit can only be written when the USART is disabled (UE=0).

Note: ONEBIT feature applies only to data bits, It does not apply to Start bit.

Anmerkung: In diesem Projekt werden DEM=1 (DE-Funktion enabelt), OVERDIS=1 (Bei einem Overrun werden alte Daten überschrieben) und ONEBIT=1 (One-Sample-Bit-Methode) gesetzt.

Baud Rate Register

Auszug aus RM S. 633

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.

Anmerkung: USARTDIV beträgt in diesem Projekt 0x4E2.

Request Register

Auszug aus RM S. 634

23.7.6 Request register (USART_RQR)

Address offset: 0x18

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
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	RXFRQ	MMRQ	SBKRQ	ABRRQ
												w	w	w	w

Bits 31:4 Reserved, must be kept at reset value

Bit 3 **RXFRQ**: Receive data flush request
Writing 1 to this bit clears the RXNE flag.
This allows to discard the received data without reading it, and avoid an overrun condition.

Anmerkung: RXFRQ wird in der Interrupt-Service-Routine verwendet, um das RXNE-Flag (dieses wird beim Empfang von Daten gesetzt) zurückzusetzen.

Interrupt And Status Register

Auszug aus RM S. 635-638

23.7.7 Interrupt and status register (USART_ISR)

Address offset: 0x1C

Reset value: 0x0200 00C0

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.	RWU	SBKF	CMF	BUSY
												r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ABRF	ABRE	Res.	Res.	RTOF	CTS	CTSIF	Res.	TXE	TC	RXNE	IDLE	ORE	NF	FE	PE
r	r			r	r	r		r	r	r	r	r	r	r	r

Bit 7 **TXE**: Transmit data register empty
This bit is set by hardware when the content of the USART_TDR register has been transferred into the shift register. It is cleared by a write to the USART_TDR register.
An interrupt is generated if the TXEIE bit =1 in the USART_CR1 register.
0: data is not transferred to the shift register
1: data is transferred to the shift register)
Note: This bit is used during single buffer transmission.

Bit 6 **TC**: Transmission complete

This bit is set by hardware if the transmission of a frame containing data is complete and if TXE is set. An interrupt is generated if TCIE=1 in the USART_CR1 register. It is cleared by software, writing 1 to the TCCF in the USART_ICR register or by a write to the USART_TDR register.
An interrupt is generated if TCIE=1 in the USART_CR1 register.
0: Transmission is not complete
1: Transmission is complete

Note: If TE bit is reset and no transmission is on going, the TC bit will be set immediately.

Bit 5 **RXNE**: Read data register not empty

This bit is set by hardware when the content of the RDR shift register has been transferred to the USART_RDR register. It is cleared by a read to the USART_RDR register. The RXNE flag can also be cleared by writing 1 to the RXFRQ in the USART_RQR register.
An interrupt is generated if RXNEIE=1 in the USART_CR1 register.
0: data is not received
1: Received data is ready to be read.

Anmerkung TXE wird benötigt, um den Zeitpunkt zu bestimmen, zu welchem neue Daten in das TDR-Register geladen werden können. TC wird beim Ende des Sendevorgangs benötigt, RXNE wird beim Empfang von Daten gesetzt und wird daher verwendet, um jene Interrupt-Service-Routine auszulösen, welche ihrerseits den Sendevorgang zum Übertragen des Inhalts des 10-Byte-Buffers gemäß Aufgabenstellung auslöst.

Receive Data Register

Auszug aus RM S. 639

23.7.9 Receive data register (USART_RDR)

Address offset: 0x24

Reset value: Undefined

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
Res.	Res.	Res.	Res.	Res.	Res.	Res.	RDR[8:0]								
							r	r	r	r	r	r	r	r	r

Bits 31:9 Reserved, must be kept at reset value.

Bits 8:0 **RDR[8:0]**: Receive data value

Contains the received data character.
The RDR register provides the parallel interface between the input shift register and the internal bus (see *Figure 226*).
When receiving with the parity enabled, the value read in the MSB bit is the received parity bit.

Transmit Data Register

Auszug aus RM S. 640

23.7.10 Transmit data register (USART_TDR)

Address offset: 0x28

Reset value: Undefined

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
Res.	Res.	Res.	Res.	Res.	Res.	Res.	TDR[8:0]								
							rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:9 Reserved, must be kept at reset value.

Bits 8:0 **TDR[8:0]**: Transmit data value

Contains the data character to be transmitted.

The TDR register provides the parallel interface between the internal bus and the output shift register (see *Figure 226*).

When transmitting with the parity enabled (PCE bit set to 1 in the USART_CR1 register), the value written in the MSB (bit 7 or bit 8 depending on the data length) has no effect because it is replaced by the parity.

Note: This register must be written only when TXE=1.

2.7.3 Implementierung in C

```
// --- USART1 configuration ---
// USART1 boundary start address: 0x40013800

// reception procedure
uint32_t *usart_cr1; // USART control register 1
usart_cr1 = 0x40013800 + 0x00;
*usart_cr1 |= 0x1021 0620 // Word length = 9 bits (8 data
    bits + 1 parity bit), Driver Enable assertion time =
    00001, Driver Enable de-assertion time = 00001,
    oversampling rate = 16, enable parity, ODD parity
    gewaehlt, Interrupt-Enable fuer RXNE -> Interrupt bei
    Empfang von Daten

// miscellaneous
uint32_t *usart_cr3; // USART control register 3
usart_cr3 = 0x40013800 + 0x08;
*usart_cr3 |= 0x00005800 // Disable Overrun-Error, Enable
    DE-function, one sample bit method
```

```
// Configuration baud rate
uint32_t *usart_brr; // Baud rate register
usart_brr = 0x40013800 + 0x0C;
*usart_brr |= 0x000004E2; // Baud rate = 38400 fuer f=48MHz

// configuration of the amount of stop bits
// Default: 1 stop bit

usart_cr1 |= 0x00000001; // Enable USART1

usart_cr1 |= 0x00000004; // Set the RE bit for enabling
   reception
```

2.8 Konfiguration des NVIC und Implementierung der Interrupt-Service-Routinen

siehe RM S. 170/171

2.8.1 Allgemeines

NVIC

Sämtliche Interrupts werden vom Nested Vectored Interrupt Controller gehandhabt. Die Vektor-Tabelle ist im RM S. 170/171 zu finden:

Table 32. Vector table

Position	Priority	Type of priority	Acronym	Description	Address
12	19	settable	ADC	ADC interrupts	0x0000 0070
27	34	settable	USART1	USART1 global interrupt	0x0000 00AC

Für die Initialisierung des NVICs gibt es im RM auf S. 736 ein Code-Beispiel:

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) */
```

Die dafür verwendeten Funktionen werden im PM0215 programming manual auf S. 70 beschrieben:

Table 26. CMSIS access NVIC functions

CMSIS function(1)	Description
void NVIC_EnableIRQ(IRQn_Type IRQn)	Enables an interrupt or exception.
void NVIC_DisableIRQ(IRQn_Type IRQn)	Disables an interrupt or exception.
void NVIC_SetPendingIRQ(IRQn_Type IRQn)	Sets pending status of interrupt or exception to 1.
void NVIC_ClearPendingIRQ(IRQn_Type IRQn)	Clears pending status of interrupt / exception to 0.
uint32_t NVIC_GetPendingIRQ(IRQn_Type IRQn)	Reads the pending status of interrupt / exception. Returns non-zero value if pending status is set to 1.
void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority)	Sets priority of an interrupt / exception with configurable priority level to 1.
uint32_t NVIC_GetPriority(IRQn_Type IRQn)	Reads priority of an interrupt or exception with configurable priority level. Returns the current priority level.

1. The input parameter IRQn is the IRQ number,

Die Einträge der Vekor-Tabelle beinhalten die Position, die Priorität, die Beschreibung des entsprechenden Interrupts und die Adresse.

Die Position wird sowohl beim Enabeln als auch beim Setzen der Priorität mithilfe der CMSIS-Funktionen void NVIC_EnableIRQ(IRQn_Type IRQn) und void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority) benötigt; der für IRQn einzutragende Wert entspricht der Position des Interrupts.

Die Priorität legt fest, was geschieht, wenn mehrere Interrupt-Ereignisse gleichzeitig auftreten oder wenn ein Interrupt-Ereignis auftritt, wenn ein anderer gerade bearbeitet wird - in solchen Fällen gehen Vektoren mit einem niedrigeren Prioritätswert vor, bzw. unterbrechen die Ausführung der Interrupt-Service-Routine des anderen Interrupt-Ereignisses.
Die drei höchstpriorisierten Interrupts sind der Reset (Priorität=-3), der nicht-maskierbare Interrupt NMI (Interrupt=-2) und der Hardfault-Interupt (Priorität=-1). Die Priorität aller anderen Interrupts kann mittels der Funktion void nVIC_SetPriority(IRQn_Type IRQn, uint32_t priority) eingestellt werden, wobei IRQn der Position und priority der einzustellenden Priorität des jeweiligen Interrupts als Parameter übergeben werden. Zwei Bits werden für die Priorität des Interrupts verwendet, weshalb zwischen vier verschiedenen priority levels unterschieden werden kann.

Die im Eintrag eines Interrupts angegebene Adresse enthält die Startadresse

der zugehörigen Interrupt-Service-Routine. Da sich diese im Flash befindet und dieser vor Programmstart beschrieben werden muss, muss diese mittels eines Linker-Scripts beschrieben werden. Dafür gibt es für den STM32 bereits vorgefertigte Skripte für verschiedene Toolchains, dieses `https://github.com/pichenettes/stmlib/blob/master/third_party/STM/CMSIS/CM3_f0xx/startup/arm/startup_stm32f030.s` ist für die MDK-ARM-Toolchain konzipiert. Damit eine Funktion dem entsprechenden Interrupt als Interrupt-Service-Routine zugewiesen wird, muss der Name der Funktion dem entsprechenden Eintrag im vorher verlinkten Script entsprechen. Die Bezeichnungen für die Interrupt-Service-Routinen des ADC und der USART1 lauten demnach ADC1_IRQHandler und USART1_IRQHandler.

Interrupt-Service-Routinen

Als Interrupt-Service-Routinen werden Funktionen bezeichnet, welche bei einem Interrupt-Ereignis aufgerufen werden. Bei einem solchen Aufruf wird zunächst der Programm-Kontext - der Inhalt mehrerer Register, darunter auch die Rücksprung-Adresse - am Stack gespeichert und der Stack-Pointer dekrementiert (Dies wird als Push-Operation bezeichnet). Nach dem Abhandeln der Interrupt-Service-Routine wird anschließend der vorherige Progamm-Kontext durch eine Pop-Operation wiederhergestellt, indem die Daten am Stack eingelesen werden (die Rücksprung-Adresse wird dabei in den Program Counter geladen) und anschließend der Stack-Pointer inkrementiert wird.
Da Interrupt-Service-Routinen das Hauptprogramm unterbrechen, sollten diese möglichst kurz sein, weil ansonsten im schlechtesten Fall noch vor Beendigung derselben erneut dasselbe Interrupt-Ereignis auftritt, was dazu führen würde, dass das Hauptprogramm nicht ausgeführt werden könnte. Weiters ist zu beachten, dass in der Interrupt-Service-Routine jenes Flag, welches diese ausgelöst hatte, zurückgesetzt wird, weil ansonsten diese sofort erneut ausgeführt werden würden.
Zuletzt sei angemerkt, dass das Schlüsselwort 'volatile' Variablen in C vor Optimierung schützt; dies ist vor allem bei der Programmierung von Interrupt-Service-Routinen notwendig, weil in diesen oft Flags gesetzt werden, auf welche im Hauptptrogramm geprüft wird. Wenn sich diese Variablen lediglich in der Interrupt-Service-Routine ändern, so hat dies zur Folge, dass viele C-Optimierer diese entfernen, was dazu führt, dass die von Interrupts abhängigen Code-Teile nie ausgeführt würden. Dies wird durch die Verwendung des Schlüsselworts 'volatile' verhindert.

2.8.2 Implementierung in C

```
volatile int adc_interrupt_flag = 0; // dient der
    Ueberpruefung, ob ein neues Wandelergebnis vorliegt
volatile int usart_interrupt_flag = 0; // dient der
    Ueberpruefung, ob Daten empfangen worden sind

// ADC-Interrupt-Service-Routine
// ADC boundary start address: 0x40012400
void ADC1_IRQHandler(void) __attribute__((interrupt));
void ADC1_IRQHandler(void){
        uint32_t *adc_isr; // Interrupt And Status Register
        adc_isr = 0x40012400 + 0x00;
        *adc_isr |= 0x00000004; // Clear EOC-flag
        adc_interupt_flag = 1; // Wandelergebnis liegt vor
}

// USART1-Interrupt-Service-Routine
// USART1 boundary start address: 0x40013800
void USART1_IRQHandler(void __attribute__((interrupt));
void USART1_IRQHandler(void){
        uint32_t *usart_rqr; // Request Register
        usart_rqr = 0x40013800 + 0x18;
        *usart_rqr |= 0x00000008; // Clear RXNE-flag
        usart_interrupt_flag = 1; // Daten wurden empfangen
}

// --- NVIC/Interrupt configuration ---
NVIC_EnableIRQ(12); // Enable ADC-Interrupt
NVIC_SetPriority(12, 3); // Setzen der Prioritaet

NVIC_EnableIRQ(27); // Enable USART1-Interrupt
NVIC_SetPriority(27, 2); // Setzen der Prioritaet
```

2.9 Implementierung in C

```
/*
Titel: main
Beschreibung:
Autor: Patrick Wintner
GitHub: https://github.com/EternalNothingness/https://
    github.com/EternalNothingness/DIC-5BHEL-Projekt_1-
    Serielle-Kommunikation-mit-uC.git.git
```

```
Datum der letzten Bearbeitung: 25.01.2021
*/

#include <stdlib.h>
#include <stdint.h>

volatile int adc_interrupt_flag = 0; // dient der
    Ueberpruefung, ob ein neues Wandelergebnis vorliegt
volatile int usart_interrupt_flag = 0; // dient der
    Ueberpruefung, ob Daten empfangen worden sind

// ADC-Interrupt-Service-Routine
// ADC boundary start address: 0x40012400
void ADC1_IRQHandler(void) __attribute__((interrupt));
void ADC1_IRQHandler(void){
        uint32_t *adc_isr; // Interrupt And Status Register
        adc_isr = 0x40012400 + 0x00;
        *adc_isr |= 0x00000004; // Clear EOC-flag
        adc_interupt_flag = 1; // Wandelergebnis liegt vor
}

// USART1-Interrupt-Service-Routine
// USART1 boundary start address: 0x40013800
void USART1_IRQHandler(void __attribute__((interrupt));
void USART1_IRQHandler(void){
        uint32_t *usart_rqr; // Request Register
        usart_rqr = 0x40013800 + 0x18;
        *usart_rqr |= 0x00000008; // Clear RXNE-flag
        usart_interrupt_flag = 1; // Daten wurden empfangen
}

int main(){
        // -- boot configuration --
        // Default; Boot von Main Flash Memory (Boot0 = 0)
        // ------------------------------------------------

        // -- alternate function configuration --
        // Port A boundary start address: 0x48000000

        uint32_t *gpioA_afrh; // GPIO alternate function
            high register
        gpioA_afrh = 0x48000000 + 0x24;
        *gpioA_afrh |= 0x00010110; // Auswahl der AF1 (
            USART1) auf PA9 (TX), PA10 (RX) und PA12 (DE)
```

```
uint32_t *gpioA_ospeedr; // GPIO port output speed
    register
gpioA_ospeedr = 0x48000000 + 0x08;
*gpioA_ospeedr |= 0x030C0000; // Auswahl High Speed
     auf PA9 (TX) und PA12 (DE)

uint32_t *gpioA_moder; // GPIO port mode register
gpioA_moder = 0x48000000 + 0x00;
*gpioA_moder |= 0x2A280003; // Enable der alternate
     functions USART1 auf PA9 (TX), PA10 (RX) und
    PA12(DE) sowie Aktivierung analog function auf
    PA0 fuer den ADC
// ------------------------------------------------

// -- clock configuration --
// RCC boundary start address: 0x4002100

uint32_t *rcc_cfgr; // Clock configuration register
rcc_cfgr = 0x4002100 + 0x04;
*rcc_cfgr |= 0x00280000; // Festlegung des
    Multiplikationsfaktors der PLL (12 => f=12*8MHz
    /2=48MHz)

uint32_t *rcc_cr; // Clock control register
rcc_cr = 0x4002100 + 0x00;
*rcc_cr |= 0x01000000; // Enable PLL
while((*rcc_cr) & 0x02000000) != 0x02000000){ //
    Warten auf PLLRDY = 1
}

*rcc_cfgr |= 0x00000002; // Festlegung der PLL als
    SYSCLK

uint32_t *rcc_cfgr3; // Clock configuration
    register 3
rcc_cfgr3 = 0x40021000 + 0x30;
*rcc_cfgr3 |= 0x00000001; // SYSCLK (=PLL) als
    USART1-CLK ausgewaehlt

uint32_t *rcc_cr2; // Clock control register 2
rcc_cr2 = 0x40021000 + 0x34;
*rcc_cr2 |= 0x00000001; // Enable HSI14

// Konfiguration ADC-Clock
// Verwendung der Default-Einstellung (async. clock
```

```
    mode -> HSI14-Clock)

uint32_t *rcc_apb2enr; // APB peripheral clock
    enable register 2
rcc_apb2enr = 0x40021000 + 0x18;
*rcc_apb2enr |= 0x00004200 // Enable CLK der USART1
    und des ADC
// ----------------------------------------------

// -- ADC configuration --
// ADC boundary start adress: 0x40012400

// Calibration software procedure
// Ensure that ADEN=0 and DMAEN=0 -> automatisch
    nach Reset der Fall
uint32_t *adc_cr; // ADC control register
adc_cr = 0x40012400 + 0x08;
*adc_cr |= 0x80000000; // Set ADCAL=1
while((*adc_cr) & 0x80000000 == 0x80000000){ //
    Wait until ADCAL=0
}

// Enable the ADC
uint32_t *adc_isr; // ADC interrupt and status
    register
adc_isr = 0x040012400 + 0x00;
*adc_isr |= 0x00000001; // Clear ADRDY bit
*adc_cr |= 0x00000001; // Set ADEN=1
while(*adc_isr == 0x00000001){// Wait until ADRDY=1
}

// miscellaneous
uint32_t *adc_cfgr1; // ADC configuration register
    1
adc_cfgr1 = 0x40012400 + 0x0C;
*adc_cfgr1 |= 0x00003010; // continuous mode,
    overrun mode (overwrite), data alignment (right)
    , resolution (8 bits)

// Sampling time selection
Defaulteinstellungen: sampling time = 1,5 clock
    cycles

// Channel Selection
uint32_t *adc_chselr; // ADC channel selection
```

```
    register
adc_chselr = 0x40012400 + 0x28;
*adc_chselr |= 0x00000001; // Channel 0 ausgewaehlt

// Enable Interrupts
uint32_t *adc_ier; // ADC interrupt enable register
adc_ier = 0x40012400 + 0x04;
*adc_ier |= 0x00000004; // Interrupt bei Vorliegen
    des Wandelergebnisses

// Set ADSTART
*adc_cr |= 0x00000004;
// ------------------------------------------------

// --- NVIC/Interrupt configuration ---
NVIC_EnableIRQ(12); // Enable ADC-Interrupt
NVIC_SetPriority(12, 3); // Setzen der Prioritaet

NVIC_EnableIRQ(27); // Enable USART1-Interrupt
NVIC_SetPriority(27, 2); // Setzen der Prioritaet
// ------------------------------------------------

// --- USART1 configuration ---
// USART1 boundary start address: 0x40013800

// reception procedure
uint32_t *usart_cr1; // USART control register 1
usart_cr1 = 0x40013800 + 0x00;
*usart_cr1 |= 0x1021 0620 // Word length = 9 bits
    (8 data bits + 1 parity bit), Driver Enable
    assertion time = 00001, Driver Enable de-
    assertion time = 00001, oversampling rate = 16,
    enable parity, ODD parity gewaehlt, Interrupt-
    Enable fuer RXNE -> Interrupt bei Empfang von
    Daten

// miscellaneous
uint32_t *usart_cr3; // USART control register 3
usart_cr3 = 0x40013800 + 0x08;
*usart_cr3 |= 0x00005800 // Disable Overrun-Error,
    Enable DE-function, one sample bit method

// Configuration baud rate
uint32_t *usart_brr; // Baud rate register
usart_brr = 0x40013800 + 0x0C;
```

```
*usart_brr |= 0x000004E2; // Baud rate = 38400 fuer
    f=48MHz

// configuration of the amount of stop bits
// Default: 1 stop bit

usart_cr1 |= 0x00000001; // Enable USART1

usart_cr1 |= 0x00000004; // Set the RE bit for
   enabling reception
// ----------------------------------------

// -- Other Setup Code
// adc boundary start address: 0x40012400
// usart boundary start address: 0x40013800

uint32_t *adc_dr; // ADC data register
adc_dr = 0x40012400 + 0x40;

uint32 *usart_isr; // USART interrupt and status
   register
usart_isr = 0x40013800 + 0x1C;
uint32 *usart_tdr; // USART transmit data register
usart_tdr = 0x40013800 + 0x28;

uint32_t *buf=malloc(10*sizeof(uint32_t)); // 10
   Byte (nur die jeweils letzten 8 Bit werden
   verwendet) Buffer fuer ADC-Wandel-Ergebnisse

for(int i=0; i<=9; i++){
        *(buf+i) = 0x00; // Reset buffer contents
}
// ----------------------------------------

// -- main loop --
for(;;){
        if(adc_interrupt_flag == 1){
                for(int i=9; i>0; i--){
                        *(buf+i) = *(buf+i-1); //
                           shift right
                }
                *buf = *adc_dr; // latest data
                   written at the beginning of the
                   buffer
                *adc_interrupt_flag = 0; // Reset
```

```
                    Interrupt-Flag
                }
                if(usart_interrupt_flag == 1){
                        *adc_crr |= 0x00000010// Stop ADC
                            to avoid interruptions during
                            transmission
                        *usart_cr1 &= 0xFFFFFFFB // Disable
                             Reception
                        *usart_cr1 |= 0x00000008 // Enable
                            Transmission
                        for(int i=9; i>=0; i--){
                                *usart_tdr = *(buf+i); //
                                    Older data is sent first
                                while((usart_isr & 0
                                    x00000080) != 0x00000080
                                    ){ // Wait until data is
                                     transferred to the
                                    shift register
                                }
                        }
                        while((usart_isr & 0x00000040) != 0
                            x00000040){ // Wait until
                            transmission complete
                        }
                        *usart_cr1 &= 0xFFFFFFF7 // Disable
                             Transmission
                        *usart_cr1 |= 0x00000004; // Enable
                             Reception
                        *adc_cr |= 0x00000004; // Start ADC
                        *usart_interrupt_flag = 0; // Reset
                             Interrupt-Flag
                }
        }
}
```