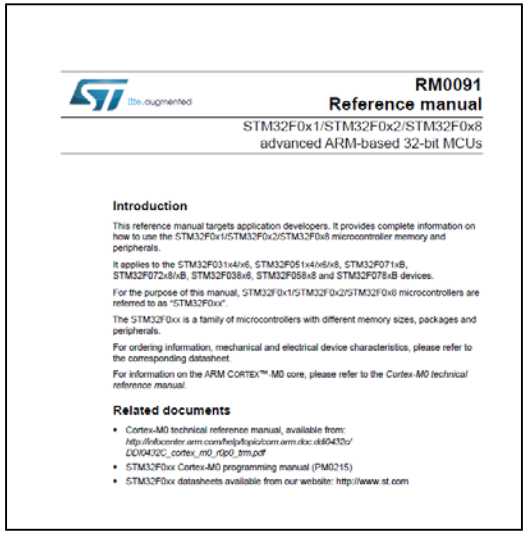
Embedded Communication

UART and RS-232

Chapter 8 Universal Asynchronous Serial Communication
"Mastering STM32" by Carmine Noviello
https://leanpub.com/mastering-stm32

Chapter 26 Universal Synchronous Asynchronous Receiver Transmitter (USART)
RM0091: STM32F0x1 Reference manual


RM0091
Reference manual
STM32F0x1/STM32F0x2/STM32F0x8
advanced ARM-based 32-bit MCUs

Introduction

This reference manual targets application developers. It provides complete information on how to use the STM32F0x1/STM32F0x2/STM32F0x8 microcontroller memory and peripherals.

It applies to the STM32F031x4/x6, STM32F051x4/x6/x8, STM32F071xB, STM32F072x8/xB, STM32F038x8, STM32F058x8 and STM32F078xB devices.

For the purpose of this manual, STM32F0x1/STM32F0x2/STM32F0x8 microcontrollers are referred to as "STM32F0xx".

The STM32F0xx is a family of microcontrollers with different memory sizes, packages and peripherals.

For ordering information, mechanical and electrical device characteristics, please refer to the corresponding datasheet.

For information on the ARM CORTEX™-M0 core, please refer to the *Cortex-M0 technical reference manual*.

Related documents

- Cortex-M0 technical reference manual, available from: http://infocenter.arm.com/help/topic/com.arm.doc.ddi0432c/DDI0432C_cortex_m0_r0p0_trm.pdf
- STM32F0xx Cortex-M0 programming manual (PM0215)
- STM32F0xx datasheets available from our website: http://www.st.com

UART and RS-232
Implementation on the STM32F0

UART and the RS-232
STM32F0

- STM32F051 has two USART hardware modules
  - USART 1
  - USART 2

UART and the RS-232
STM32F0

- STM32F051 has two USART hardware modules
  - USART 1
  - USART 2

- Message structure configurability
  - Length of data: 8-bits or 9 bits (including the parity bit)
  - Order of data: LSB first or MSB first
  - Baud rate: 1.2kbps, 2.4kbps, 4.8kbps, 9.6kbps, 14.4kbps, …
  - Number of stop bits: 1 or 2 bits
  - Option to include or exclude parity bit in the message structure

UART and the RS-232
STM32F0

- Pins on the STM32F0 that can be used and their AF
  - PA14 (AF1) – USART2_TX
  - PA15 (AF1) – USART2_RX

  - PA2 (AF1) – USART2_TX
  - PA3 (AF1) – USART2_RX

  - PA9 (AF1) – USART1_TX
  - PA10 (AF1) – USART1_RX

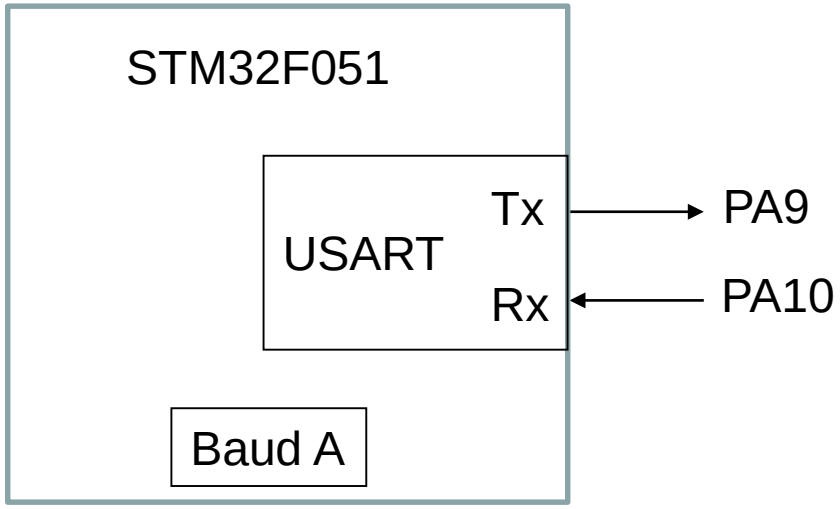
  - PB6 (AF0) – USART1_TX
  - PB7 (AF0) – USART1_RX

UART and RS-232

Steps to program the STM32F0

UART and the RS-232
Steps to program the STM32F051

- **Objective**
  - Baud Rate: 115.2kbps
  - Data bits: 8
  - No parity
  - Number of stop bits: 1
- **Assumptions**
  - f_{CLK} = 48MHz



UART and the RS-232
Steps for intialisation: STM32F051

- Step 1: Enable clock on GPIO port A and USART1
  - Set IOPAEN (bit 17) in RCC_AHBENR to enable PORT A

```
RCC -> AHBENR |= RCC_AHBENR_GPIOAEN;          // enable clock PORT A
```

Bit 17 **IOPAEN:** I/O port A clock enable
Set and cleared by software.
0: I/O port A clock disabled
1: I/O port A clock enabled

7.4.6 AHB peripheral clock enable register (RCC_AHBENR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	TSCEN	Res.	IOPF EN	IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.
							rw		rw	rw	rw	rw	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.	CRC EN	Res.	FLITF EN	Res.	SRAM EN	Res.	DMA EN
									rw		rw		rw		rw

Extracted from the STM32F051 reference manual

UART and the RS-232
Steps for intialisation: STM32F051

- Step 1: Enable clock on GPIO port A and USART1
  - Set IOPAEN (bit 17) in RCC_AHBENR to enable PORT A
  - Set USART1_EN (bit 14) in RCC_APB2ENR

```
RCC -> AHBENR |= RCC_AHBENR_GPIOAEN;        // enable clock PORT A
RCC -> APB2ENR |= RCC_APB2ENR_USART1EN;     // enable clock for USART 1
```

Bit 14 **USART1EN:** USART1 clock enable
0: USART1clock disabled
1: USART1clock enabled

7.4.7 APB peripheral clock enable register 2 (RCC_APB2ENR)

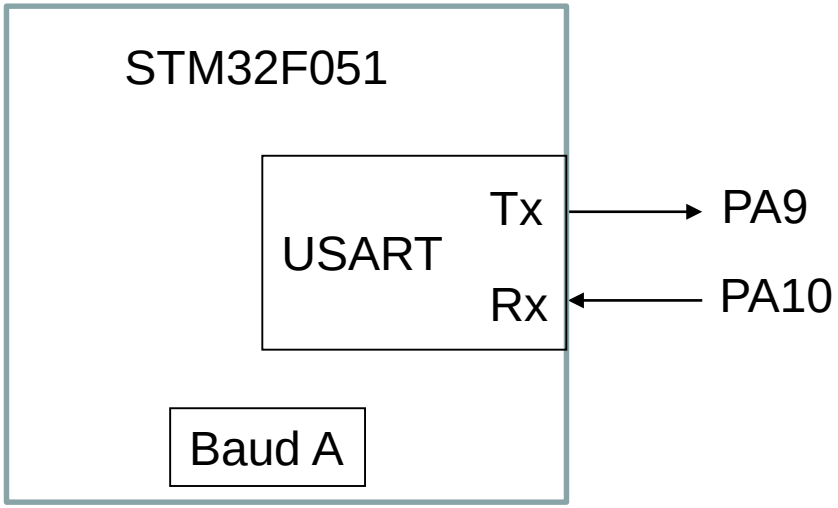
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	DBGM CUEN	Res.	Res.	Res.	TIM17 EN	TIM16 EN	TIM15 EN
									rw				rw	rw	rw

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	USART1 EN	Res.	SPI1 EN	TIM1 EN	Res.	ADC EN	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	SYSCFG COMP EN
	rw		rw	rw		rw									rw

Extracted from the STM32F051 reference manual

UART and the RS-232
Steps for intialisation: STM32F051

- Step 1: Enable clock on GPIO port A and USART1
- Step 2: Configure port pin as alternative function mode
  - To configure PA9 and PA10 to AF, set MODER9[1:0] and MODER10[1:0] to ‘10’ in GPIOA_MODER



9.4.1 GPIO port mode register (GPIOx_MODER) (x = A..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

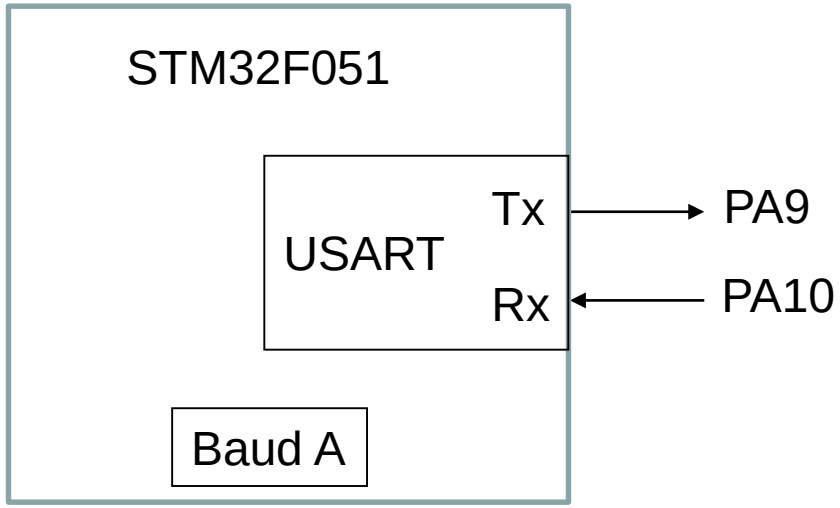
Bits 2y+1:2y **MODERy[1:0]:** Port x configuration bits (y = 0..15)
These bits are written by software to configure the I/O mode.
00: Input mode (reset state)
01: General purpose output mode
10: Alternate function mode
11: Analog mode

Extracted from the STM32F051 reference manual

UART and the RS-232
Steps for intialisation: STM32F051

- Step 1: Enable clock on GPIO port A and USART1
- Step 2: Configure port pin as alternative function mode
  - To configure PA9 and PA10 to AF, set MODER9[1:0] and MODER10[1:0] to '10' in GPIOA_MODER

```
GPIOA -> MODER |= GPIO_MODER_MODER9_1;    // set PA9 to AF mode
GPIOA -> MODER |= GPIO_MODER_MODER10_1;   // set PA10 to AF mode
```



9.4.1 GPIO port mode register (GPIOx_MODER) (x = A..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

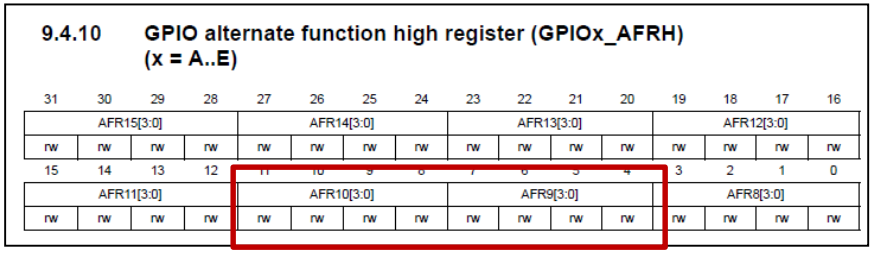
Extracted from the STM32F051 reference manual

UART and the RS-232

Steps for intialisation: STM32F051

- Step 1: Enable clock on GPIO port A and USART1
- Step 2: Configure port pin as alternative function mode
- Step 3: Map PA9 to USART1_Tx and PA10 to USART1_Rx
  - Set AFR9 to ‘0001’ in GPIOA_AFRH
  - Set ARF10 to ‘0001’ in GPIOA_AFRH

AFRy selection:
0000: AF0
0001: AF1
0010: AF2
0011: AF3



9.4.10 GPIO alternate function high register (GPIOx_AFRH) (x = A..E)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFR15[3:0]				AFR14[3:0]				AFR13[3:0]				AFR12[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
AFR11[3:0]				AFR10[3:0]				AFR9[3:0]				AFR8[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Extracted from the STM32F051 reference manual

Table 14. Alternate functions selected through GPIOA_AFR registers for port A

Pin name	AF0	AF1	AF2	AF3	AF4	AF5
PA0		USART2_CTS	TIM2_CH1_ETR	TSC_G1_IO1		
PA1	EVENTOUT	USART2_RTS	TIM2_CH2	TSC_G1_IO2		
PA2	TIM15_CH1	USART2_TX	TIM2_CH3	TSC_G1_IO3		
PA3	TIM15_CH2	USART2_RX	TIM2_CH4	TSC_G1_IO4		
PA4	SPI1_NSS, I2S1_WS	USART2_CK		TSC_G2_IO1	TIM14_CH1	
PA5	SPI1_SCK, I2S1_CK	CEC	TIM2_CH1_ETR	TSC_G2_IO2		
PA6	SPI1_MISO, I2S1_MCK	TIM3_CH1	TIM1_BKIN	TSC_G2_IO3		TIM16_CH1
PA7	SPI1_MOSI, I2S1_SD	TIM3_CH2	TIM1_CH1N	TSC_G2_IO4	TIM14_CH1	TIM17_CH1
PA8	MCO	USART1_CK	TIM1_CH1	EVENTOUT		
PA9	TIM15_BKIN	USART1_TX	TIM1_CH2	TSC_G4_IO1		
PA10	TIM17_BKIN	USART1_RX	TIM1_CH3	TSC_G4_IO2		
PA11	EVENTOUT	USART1_CTS	TIM1_CH4	TSC_G4_IO3		
PA12	EVENTOUT	USART1_RTS	TIM1_ETR	TSC_G4_IO4		
PA13	SWDIO	IR_OUT				
PA14	SWCLK	USART2_TX				
PA15	SPI1_NSS, I2S1_WS	USART2_RX	TIM2_CH1_ETR	EVENTOUT		

Extracted from the STM32F051 datasheet

UART and the RS-232
Steps for intialisation: STM32F051

- Step 1: Enable clock on GPIO port A and USART1
- Step 2: Configure port pin as alternative function mode
- Step 3: Map PA9 to USART1_Tx and PA10 to USART1_Rx
  - Set AFR9 to ‘0001’ in GPIOA_AFRH

```
# define GPIO_AFRH_PIN9_AF_0 = 0b00000000000000000000000000010000;
# define GPIO_AFRH_PIN9_AF_1 = 0b00000000000000000000000000100000;
# define GPIO_AFRH_PIN9_AF_2 = 0b00000000000000000000000001000000;
# define GPIO_AFRH_PIN9_AF_3 = 0b00000000000000000000000010000000;
GPIOA -> AFR[1] |= GPIO_AFRH_PIN9_AF_0;        // set bit 0 of AFR9[3:0]
GPIOA -> AFR[1] &= ~GPIO_AFRH_PIN9_AF_1;       // clear bit 1 of AFR9[3:0]
GPIOA -> AFR[1] &= ~GPIO_AFRH_PIN9_AF_2;       // clear bit 2 of AFR9[3:0]
GPIOA -> AFR[1] &= ~GPIO_AFRH_PIN9_AF_3;       // clear bit 3 of AFR9[3:0]
```

UART and the RS-232
Steps for intialisation: STM32F051

- Step 1: Enable clock on GPIO port A and USART1
- Step 2: Configure port pin as alternative function mode
- Step 3: Map PA9 to USART1_Tx and PA10 to USART1_Rx
  - Set AFR10 to ‘0001’ in GPIOA_AFRH

```
# define GPIO_AFRH_PIN10_AF_0 = 0b00000000000000000000000100000000;
# define GPIO_AFRH_PIN10_AF_1 = 0b00000000000000000000001000000000;
# define GPIO_AFRH_PIN10_AF_2 = 0b00000000000000000000010000000000;
# define GPIO_AFRH_PIN10_AF_3 = 0b00000000000000000000100000000000;
GPIOA -> AFR[1] |= GPIO_AFRH_PIN10_AF_0;        // set bit 0 of AFR10[3:0]
GPIOA -> AFR[1] &= ~GPIO_AFRH_PIN10_AF_1;       // clear bit 1 of AFR10[3:0]
GPIOA -> AFR[1] &= ~GPIO_AFRH_PIN10_AF_2;       // clear bit 2 of AFR10[3:0]
GPIOA -> AFR[1] &= ~GPIO_AFRH_PIN10_AF_3;       // clear bit 3 of AFR10[3:0]
```

UART and the RS-232
Steps for intialisation: STM32F051

- Step 1: Enable clock on GPIO port A and USART1
- Step 2: Configure port pin as alternative function mode
- Step 3: Map PA9 to USART1_Tx and PA10 to USART1_Rx
- Step 4: Set length of data to 8-bits
  - Clear M0 (bit 12) in USART1_CR1

Bit 12 **M0**: Word length

This bit determines the word length.

0: 1 Start bit, 8 Data bits, n Stop bit

1: 1 Start bit, 9 Data bits, n Stop bit

26.7.1 Control register 1 (USARTx_CR1)

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

Extracted from the STM32F051 reference manual

UART and the RS-232
Steps for intialisation: STM32F051

- Step 1: Enable clock on GPIO port A and USART1
- Step 2: Configure port pin as alternative function mode
- Step 3: Map PA9 to USART1_Tx and PA10 to USART1_Rx
- Step 4: Set length of data to 8-bits
  - Clear M0 (bit 12) in USART1_CR1

Bit 12 **M0**: Word length

This bit determines the word length.

0: 1 Start bit, 8 Data bits, n Stop bit

1: 1 Start bit, 9 Data bits, n Stop bit

```
USART1 -> CR1 &= ~USART_CR1_M;        // clear M0 (bit 12)  of USARTx_CR1
```

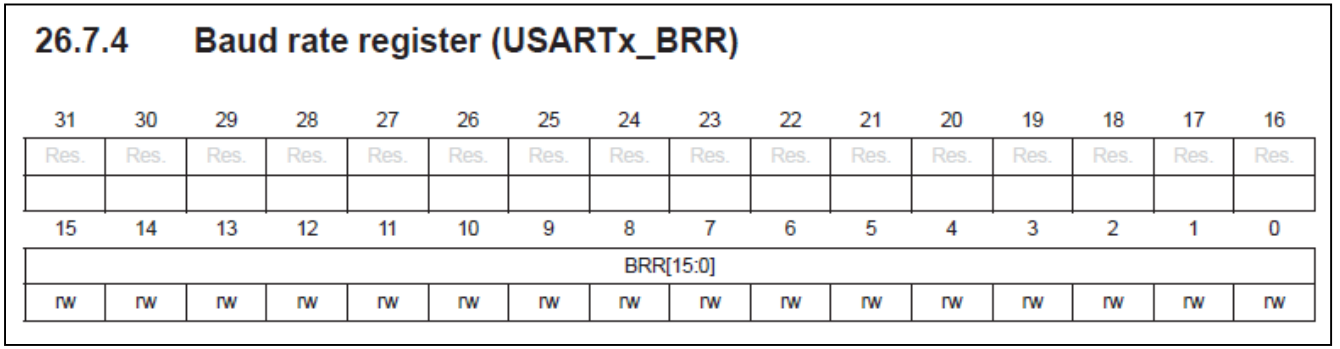
26.7.1 Control register 1 (USARTx_CR1)

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

Extracted from the STM32F051 reference manual

UART and the RS-232
Steps for intialisation: STM32F051

- Step 1: Enable clock on GPIO port A and USART1
- Step 2: Configure port pin as alternative function mode
- Step 3: Map PA9 to USART1_Tx and PA10 to USART1_Rx
- Step 4: Set length of data to 8-bits
- Step 5: Set baud rate to 115.2kbps
  - Write ‘48000000/115200’ to USART1_BRR



26.7.4 Baud rate register (USARTx_BRR)

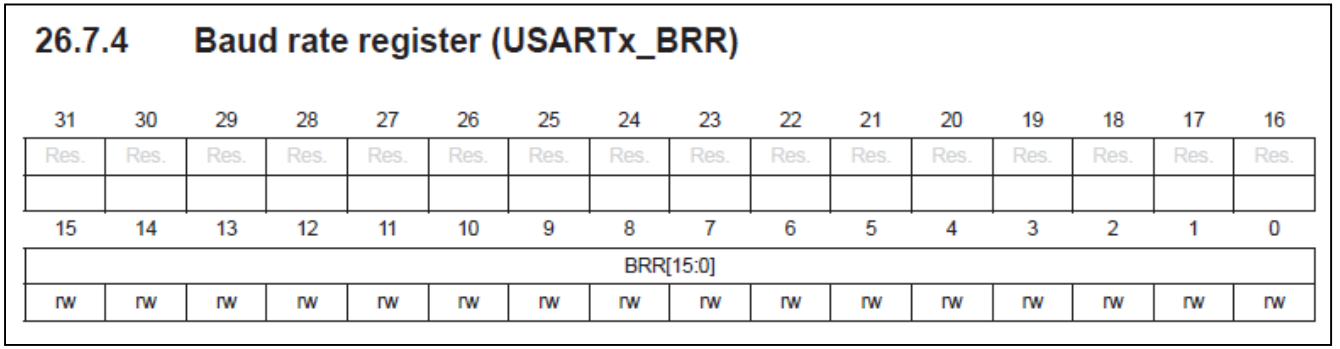
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BRR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Extracted from the STM32F051 reference manual

UART and the RS-232
Steps for intialisation: STM32F051

- Step 1: Enable clock on GPIO port A and USART1
- Step 2: Configure port pin as alternative function mode
- Step 3: Map PA9 to USART1_Tx and PA10 to USART1_Rx
- Step 4: Set length of data to 8-bits
- Step 5: Set baud rate to 115.2kbps
  - Write ‘48000000/115200’ to USART1_BRR

```
USART1 -> BRR = 48000000/115200;      // fCLK/Baud_rate sets baud rate to 115.2kbps
                // See Table 93 (page 663/914) of reference manual for more information
```



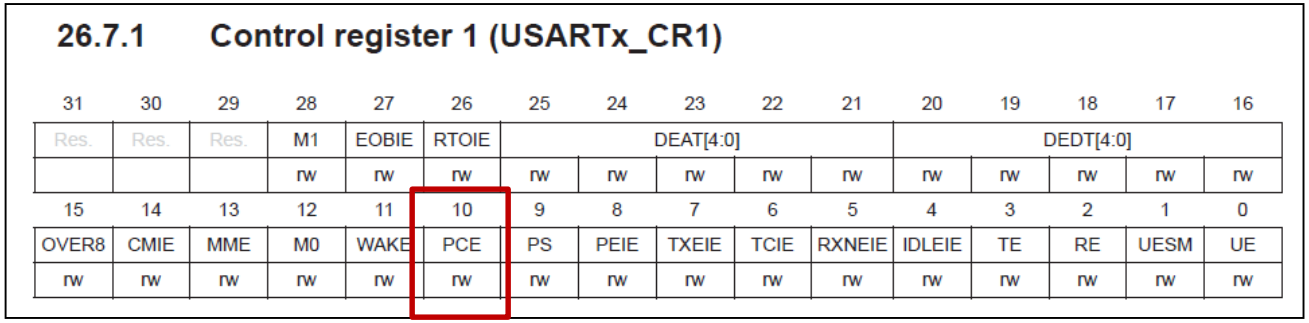
26.7.4 Baud rate register (USARTx_BRR)

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

Extracted from the STM32F051 reference manual

UART and the RS-232
Steps for intialisation: STM32F051

- Step 1: Enable clock on GPIO port A and USART1
- Step 2: Configure port pin as alternative function mode
- Step 3: Map PA9 to USART1_Tx and PA10 to USART1_Rx
- Step 4: Set length of data to 8-bits
- Step 5: Set baud rate to 115.2kbps
- Step 6: No parity
  - Clear PCE (bit 10) in USART 1_CR1



26.7.1 Control register 1 (USARTx_CR1)

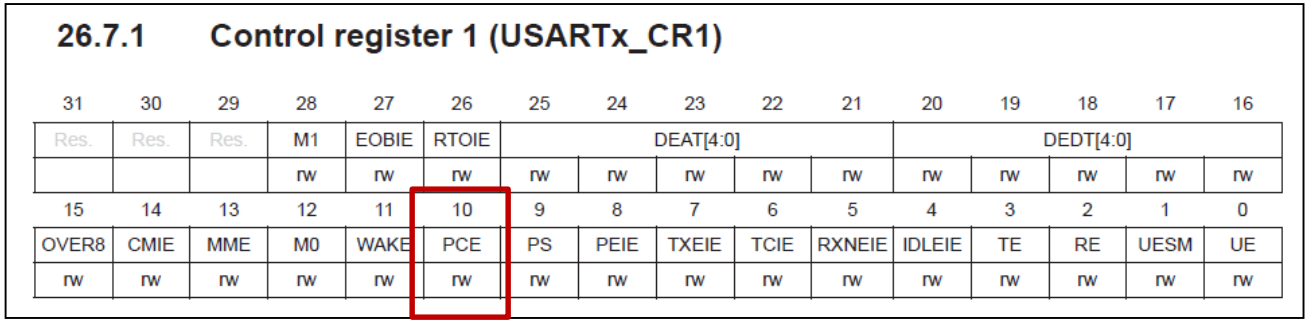
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	M1	EOBIE	RTOIE	DEAT[4:0]					DEDT[4:0]				
			rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OVER8	CMIE	MME	M0	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	UESM	UE
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Extracted from the STM32F051 reference manual

UART and the RS-232
Steps for intialisation: STM32F051

- Step 1: Enable clock on GPIO port A and USART1
- Step 2: Configure port pin as alternative function mode
- Step 3: Map PA9 to USART1_Tx and PA10 to USART1_Rx
- Step 4: Set length of data to 8-bits
- Step 5: Set baud rate to 115.2kbps
- Step 6: No parity
  - Clear PCE (bit 10) in USART 1_CR1

```
USART1 -> CR1 &= ~USART_CR1_PCE;      // no parity
```



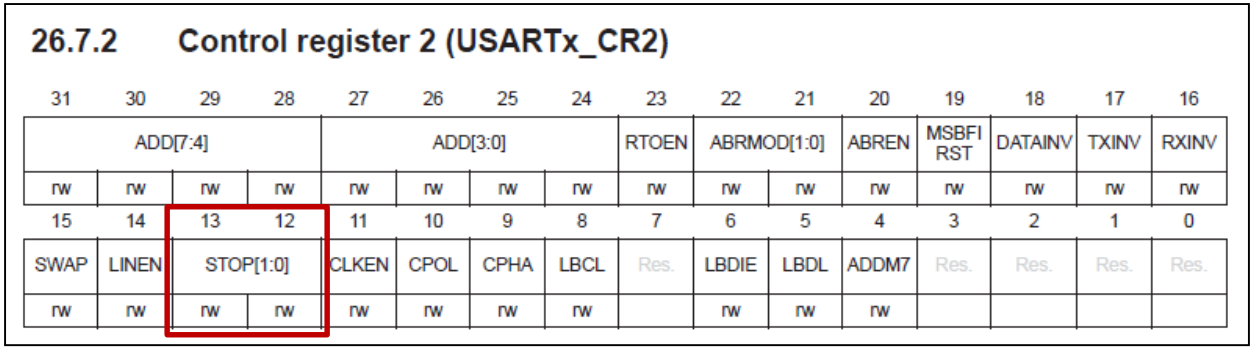
26.7.1 Control register 1 (USARTx_CR1)

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

Extracted from the STM32F051 reference manual

UART and the RS-232
Steps for intialisation: STM32F051

- Step 1: Enable clock on GPIO port A and USART1
- Step 2: Configure port pin as alternative function mode
- Step 3: Map PA9 to USART1_Tx and PA10 to USART1_Rx
- Step 4: Set length of data to 8-bits
- Step 5: Set baud rate to 115.2kbps
- Step 6: No parity
- Step 7: Set number of stop bits
  - Write ‘00’ to STOP[1:0] (bits 13:12) in USART1_CR2 for 1 stop bit



26.7.2 Control register 2 (USARTx_CR2)

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	LINEN	STOP[1:0]		CLKEN	CPOL	CPHA	LBCL	Res.	LBDIE	LBDL	ADDM7	Res.	Res.	Res.	Res.
rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw				

Extracted from the STM32F051 reference manual

UART and the RS-232
Steps for intialisation: STM32F051

- Step 1: Enable clock on GPIO port A and USART1
- Step 2: Configure port pin as alternative function mode
- Step 3: Map PA9 to USART1_Tx and PA10 to USART1_Rx
- Step 4: Set length of data to 8-bits
- Step 5: Set baud rate to 115.2kbps
- Step 6: No parity
- Step 7: Set number of stop bits
  - Write ‘00’ to STOP[1:0] (bits 13:12) in USART1_CR2 for 1 stop bit

```
USART1 -> CR2 &= ~USART_CR2_STOP;      // 1 stop bit
```

UART and the RS-232
Steps for intialisation: STM32F051

- Step 1: Enable clock on GPIO port A and USART1
- Step 2: Configure port pin as alternative function mode
- Step 3: Map PA9 to USART1_Tx and PA10 to USART1_Rx
- Step 4: Set length of data to 8-bits
- Step 5: Set baud rate to 115.2kbps
- Step 6: No parity
- Step 7: Set number of stop bits
- Step 8: Enable USART1
  - Set UE (bit 0) in USART_CR1

```
USART1 -> CR1 |= USART_CR1_UE;      // enable USART1
```

UART and the RS-232
Steps for intialisation: STM32F051

- Step 1: Enable clock on GPIO port A and USART1
- Step 2: Configure port pin as alternative function mode
- Step 3: Map PA9 to USART1_Tx and PA10 to USART1_Rx
- Step 4: Set length of data to 8-bits
- Step 5: Set baud rate to 115.2kbps
- Step 6: No parity
- Step 7: Set number of stop bits
- Step 8: Enable USART1
- Step 9: Enable USART1_TX
  - Set TE (bit 3) in USART_CR1

```
USART1 -> CR1 |= USART_CR1_TE;      // enable USART1_TX
```

UART and the RS-232
Steps for intialisation: STM32F051

- Step 1: Enable clock on GPIO port A and USART1
- Step 2: Configure port pin as alternative function mode
- Step 3: Map PA9 to USART1_Tx and PA10 to USART1_Rx
- Step 4: Set length of data to 8-bits
- Step 5: Set baud rate to 115.2kbps
- Step 6: No parity
- Step 7: Set number of stop bits
- Step 8: Enable USART1
- Step 9: Enable USART1_TX
- Step 10: Enable USART1_RX
  - Set RE (bit 2) in USART_CR1

```
USART1 -> CR1 |= USART_CR1_RE;      // enable USART1_RX
```

UART and the RS-232
C code for intialisation: STM32F051

```
void init_USART1(void)
{
 # define GPIO_AFRH_PIN9_AF_0   = 0b00000000000000000000000000010000;
 # define GPIO_AFRH_PIN9_AF_1   = 0b00000000000000000000000000100000;
 # define GPIO_AFRH_PIN9_AF_2   = 0b00000000000000000000000001000000;
 # define GPIO_AFRH_PIN9_AF_3   = 0b00000000000000000000000010000000;
 # define GPIO_AFRH_PIN10_AF_0 = 0b00000000000000000000000100000000;
 # define GPIO_AFRH_PIN10_AF_1 = 0b00000000000000000000001000000000;
 # define GPIO_AFRH_PIN10_AF_2 = 0b00000000000000000000010000000000;
 # define GPIO_AFRH_PIN10_AF_3 = 0b00000000000000000000100000000000;

RCC -> AHBENR |= RCC_AHBENR_GPIOAEN;       // enable clock PORT A
GPIOA -> MODER |= GPIO_MODER_MODER9_1;     // set PA9 to AF mode
GPIOA -> MODER |= GPIO_MODER_MODER10_1;    // set PA10 to AF mode
```

UART and the RS-232

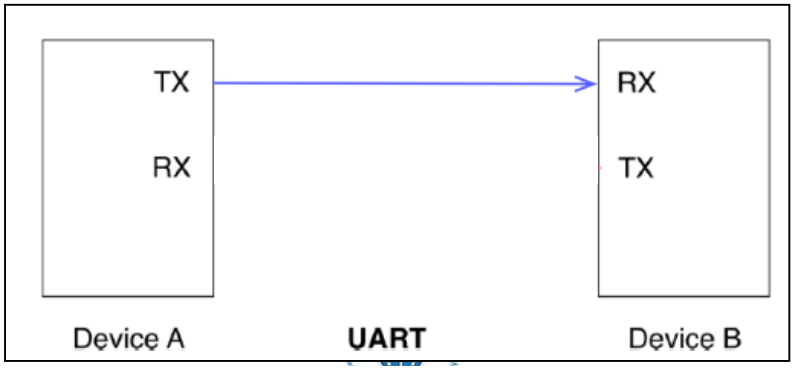
C code for intialisation: STM32F051

```
GPIOA -> AFR[1] |= GPIO_AFRH_PIN9_AF_0;        // set bit 0 of AFR9[3:0]
GPIOA -> AFR[1] &= ~GPIO_AFRH_PIN9_AF_1;     // clear bit 1 of AFR9[3:0]
GPIOA -> AFR[1] &= ~GPIO_AFRH_PIN9_AF_2;     // clear bit 2 of AFR9[3:0]
GPIOA -> AFR[1] &= ~GPIO_AFRH_PIN9_AF_3;     // clear bit 3 of AFR9[3:0]

GPIOA -> AFR[1] |= GPIO_AFRH_PIN10_AF_0;       // set bit 0 of AFR10[3:0]
GPIOA -> AFR[1] &= ~GPIO_AFRH_PIN10_AF_1;    // clear bit 1 of AFR10[3:0]
GPIOA -> AFR[1] &= ~GPIO_AFRH_PIN10_AF_2;    // clear bit 2 of AFR10[3:0]
GPIOA -> AFR[1] &= ~GPIO_AFRH_PIN10_AF_3;    // clear bit 3 of AFR10[3:0]

USART1 -> CR1 &= ~USART_CR1_M;          // clear M0 (bit 12)  of USARTx_CR1
USART1 -> CR1 |= USART_CR1_RE;           // enable USART1_RX
USART1 -> BRR = 48000000/115200;      // f_CLK/Baud_rate sets baud rate to 115.2kbps
USART1 -> CR1 &= ~USART_CR1_PCE;       // no parity
USART1 -> CR2 &= ~USART_CR2_STOP;     // 1 stop bit
USART1 -> CR1 |= USART_CR1_UE;          // enable USART1
USART1 -> CR1 |= USART_CR1_TE;          // enable USART1_TX
USART1 -> CR1 |= USART_CR1_RE;          // enable USART1_RX
}
```

UART and RS-232

Steps to program the STM32F0 (device A) to transmit data in polling mode

Device A transmits data to Device B



[taken from “Mastering STM32” by Carmine Noviello]

UART and the RS-232

Steps for transmitting data: STM32F051

- Step 1: Write data to transmit to the USART1_TDR

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

Contains the data character to be transmitted.

When transmitting with the parity enabled (PCE bit set to 1 in the USARTx_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.

26.7.11 Transmit data register (USARTx_TDR)

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

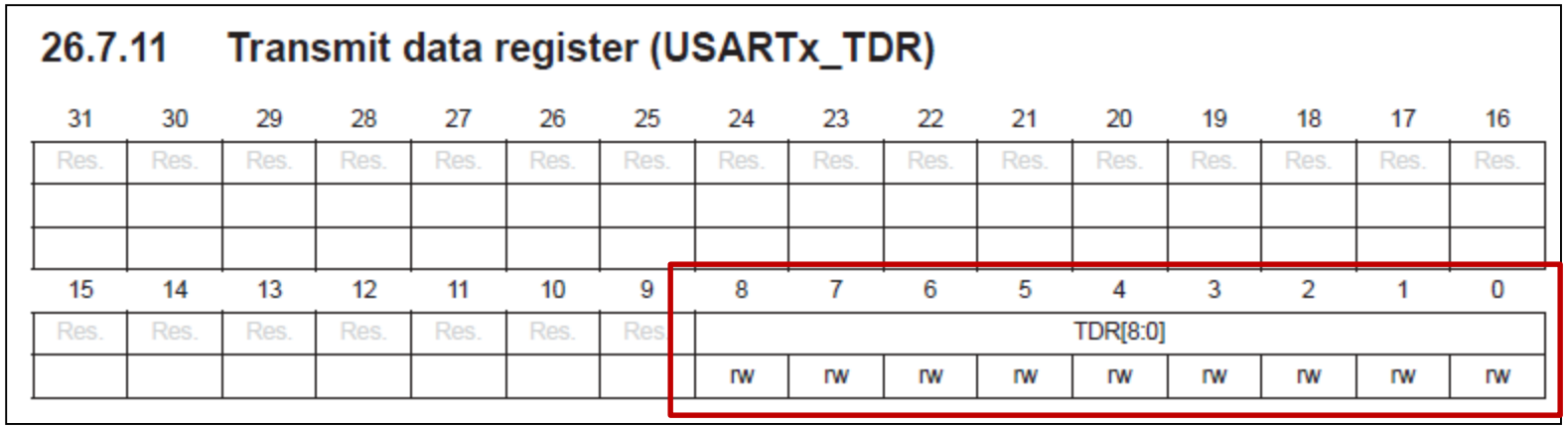
Extracted from the STM32F051 reference manual

UART and the RS-232

Steps for transmitting data: STM32F051

- Step 1: Write data to transmit to the USART1_TDR

```
unsigned char DataToTx = 'c';        // define a variable to store data to transmit
USART1 -> TDR = DataToTx;            // write character 'c' to the USART1_TDR
```



26.7.11 Transmit data register (USARTx_TDR)

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

Extracted from the STM32F051 reference manual

UART and the RS-232

Steps for transmitting data: STM32F051

- Step 1: Write data to transmit to the USART1_TDR

```
unsigned char DataToTx = 'c';          // define a variable to store data to transmit
USART1 -> TDR = DataToTx;      // write character 'c' to the USART1_TDR
```

- Step 2: Wait for data to be transmitted
  - TC (bit 6) in USART_ISR goes low when transmitting data

26.7.8 Interrupt & status register (USARTx_ISR)

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

Extracted from the STM32F051 reference manual

UART and the RS-232

Steps for transmitting data: STM32F051

- Step 1: Write data to transmit to the USART1_TDR

```
unsigned char DataToTx = 'c';          // define a variable to store data to transmit
USART1 -> TDR = DataToTx;      // write character 'c' to the USART1_TDR
```

- Step 2: Wait for data to be transmitted
  - TC (bit 6) in USART_ISR goes low when transmitting data

```
while((USART1 -> ISR & USART_ISR_TC) == 0); // exits loop when TC is high
```

26.7.8 Interrupt & status register (USARTx_ISR)

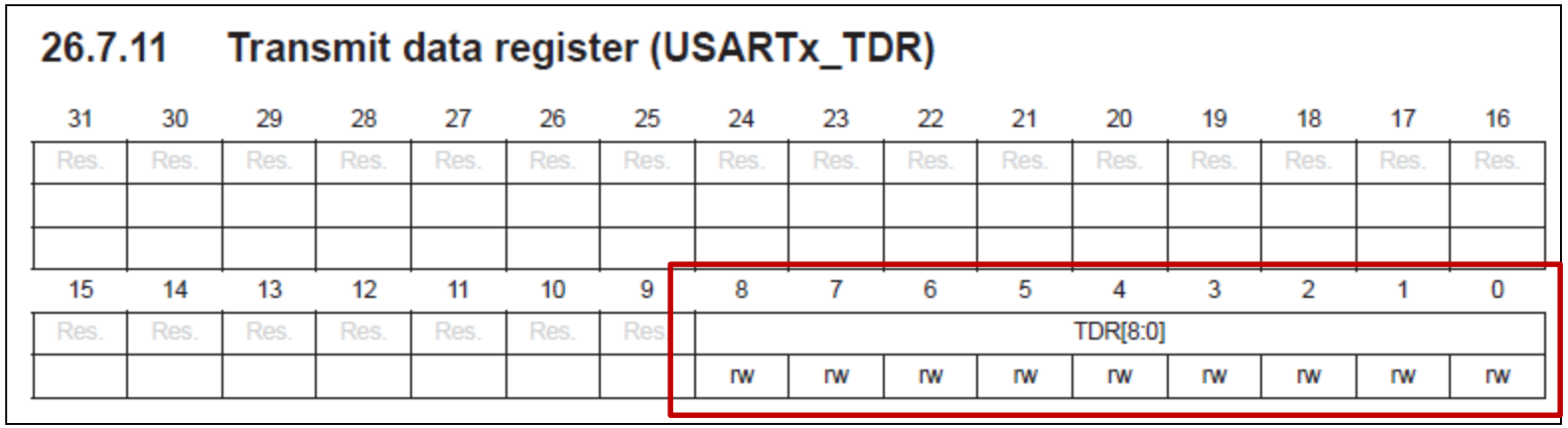
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	Res.	RE ACK	TE ACK	WUF	RWU	SBKF	CMF	BUSY
									r	r	r	r	r	r	r
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ABRF	ABRE	Res	EOBF	RTOF	CTS	CTSIF	LBDF	TXE	TC	RXNE	IDLE	ORE	NF	FE	PE
r	r		r	r	r	r	r	r	r	r	r	r	r	r	r

Extracted from the STM32F051 reference manual

UART and the RS-232

C code for transmitting data: STM32F051

```
unsigned char DataToTx = 'c';          // define a variable to store data to transmit
USART1 -> TDR = DataToTx;              // write character 'c' to the USART1_TDR
while((USART1 -> ISR & USART_ISR_TC) == 0); // wait: transmission complete
```



26.7.11 Transmit data register (USARTx_TDR)

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.

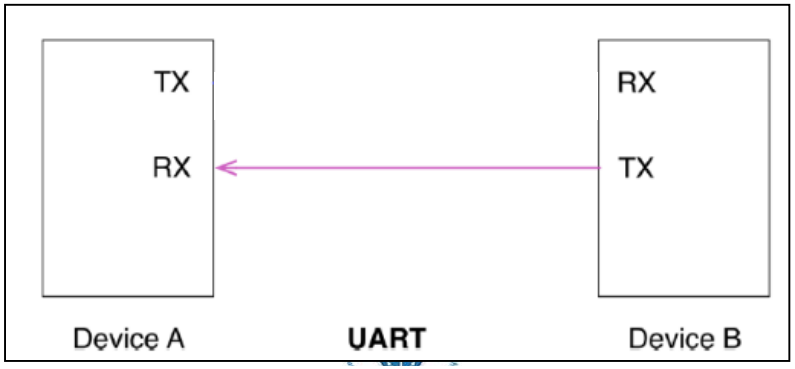
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	Res.	Res.	Res.	Res.	Res.	Res.	TDR[8:0]								
							rw	rw	rw	rw	rw	rw	rw	rw	rw

Extracted from the STM32F051 reference manual

UART and RS-232

Steps to program the STM32F0 (device A) to receive data in polling mode

Device A receives data from Device B



[taken from “Mastering STM32” by Carmine Noviello]

UART and the RS-232
Steps for receiving data: STM32F051

- Step 1: Wait for data to be received
  - When received data is ready to be read, then RXNE (bit 5) in USART_ISR goes high
    - RXNE == 0; // data not received
    - RXNE == 1; // data received

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 USARTx_RDR register. It is cleared by a read to the USARTx_RDR register. The RXNE flag can also be cleared by writing 1 to the RXFRQ in the USARTx_RQR register. An interrupt is generated if RXNEIE=1 in the USARTx_CR1 register.
0: Data is not received
1: Received data is ready to be read.

26.7.8 Interrupt & status register (USARTx_ISR)

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.	RE ACK	TE ACK	WUF	RWU	SBKF	CMF	BUSY
									r	r	r	r	r	r	r

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

Extracted from the STM32F051 reference manual

UART and the RS-232
Steps for receiving data: STM32F051

- Step 1: Wait for data to be received
  - When received data is ready to be read, then RXNE (bit 5) in USART_ISR goes high
    - RXNE == 0; // data not received
    - RXNE == 1; // data received

```
while((USART1 -> ISR & USART_ISR_RXNE) == 0); // exits loop when data is
                                              // received
```

26.7.8 Interrupt & status register (USARTx_ISR)

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.	RE ACK	TE ACK	WUF	RWU	SBKF	CMF	BUSY
									r	r	r	r	r	r	r

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

Extracted from the STM32F051 reference manual

UART and the RS-232
Steps for receiving data: STM32F051

- Step 1: Wait for data to be received
- Step 2: Store received data into a variable

```
unsigned char DataRx;          // define a variable to store data received
DataRx = USART1 -> RDR;        // store received data into 'DataRx' variable
```

26.7.10 Receive data register (USARTx_RDR)

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

Extracted from the STM32F051 reference manual

UART and the RS-232
C code for receiving data: STM32F051

- Step 1: Wait for data to be received
- Step 2: Store received data into a variable

```
unsigned char DataRx;          // define a variable to store data received

while((USART1 -> ISR & USART_ISR_RXNE) == 0); // exits loop when data is
                                              // received

DataRx = USART1 -> RDR;    // store received data into 'DataRx' variable
```