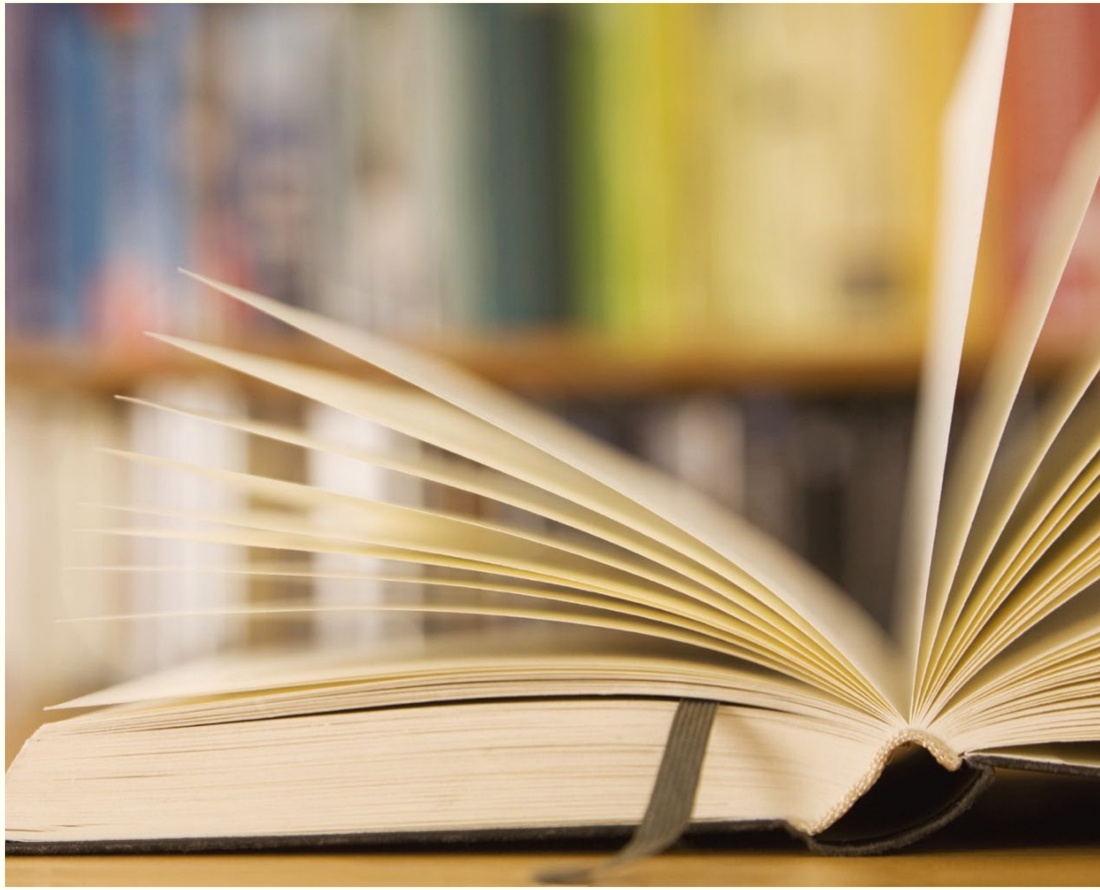
CHAPTER 8

SERIAL COMMUNICATIONS



ET1010/ET0884 MICROCONTROLLER APPLICATIONS

Author: LAN BO
School of EEE Singapore Polytechnic

8.1 Basics of serial communication

Serial vs. parallel

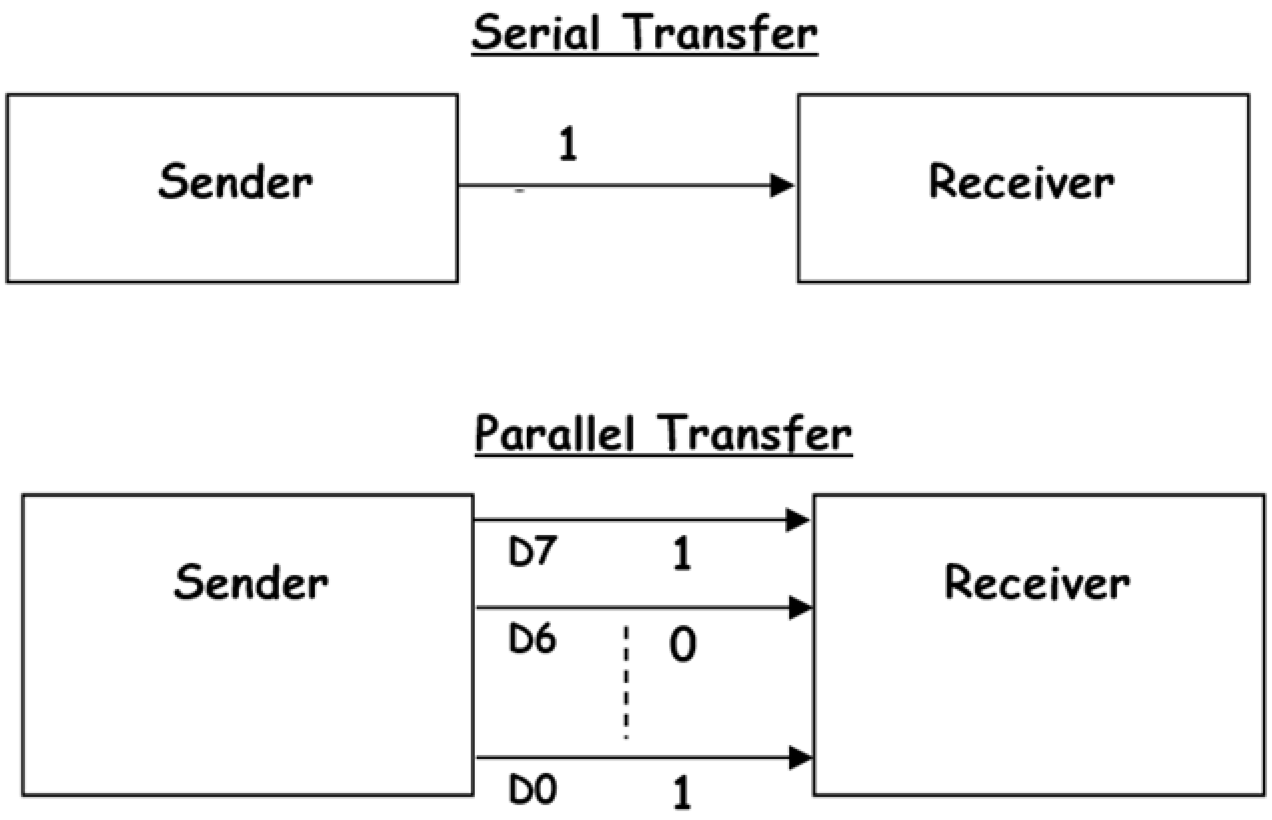


Figure 8.1 Serial vs. parallel data transfer

8.1 Basics of serial communication

- Serial data transfer is slower, but costs less as less wire is required.

- The 8-bit data that it usually handles in serial transmission must be converted using a parallel-in-serial-out shift register, before serial transmission. Additional clock cycles and algorithms may be required to handle 32-bit data in 8-bit chunks as STM32F103RB is a 32-bit microcontroller.

8.1 Basics of serial communication

Modulation / demodulation

- When the distance is short, the digital signal can be transferred as it is on a simple wire and requires no modulation, for example, data transferred between PC peripherals (e.g., keyboard) and the motherboard.
- However, for long distance data transfers using communication lines (e.g., telephone line), serial data communication requires a modem to modulate the data (i.e., convert from 0s and 1s to audio tones) before putting it on the transmission media and demodulate at the receiving end (i.e., convert from audio tones to 0s and 1s).

8.1 Basics of serial communication

Synchronous vs. Asynchronous

Serial data communication uses two methods, Synchronous and Asynchronous.

- With Synchronous communication, the clock is transmitted alongside with the data. With Asynchronous communication, no clock is transmitted.
- With Asynchronous communication, the transmitter and receiver agree on a clock speed for data transmission. They may have slight speed difference so the receiver will try to synchronize the clock to the incoming data for every character received.
- The Synchronous method usually transfers a block of data (multiple bytes/characters) at a time whereas the Asynchronous method transfers a single byte at a time.

8.1 Basics of serial communication

- Special IC chips are made to make it easier to do serial data communication. They are commonly referred to as UART (universal asynchronous receiver-transmitter) and USART (universal synchronous-asynchronous receiver-transmitter). For instance, the COM port in the PC uses an UART.

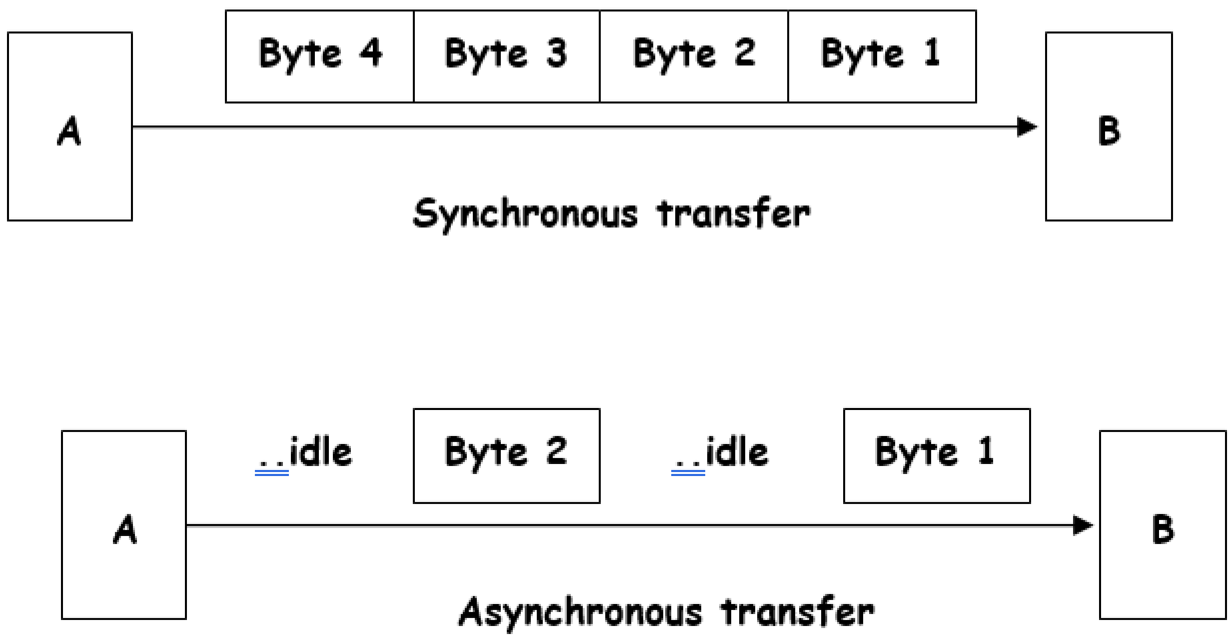


Figure 8.2 Synchronous vs. asynchronous data transfer

8.1 Basics of serial commu

Simplex, Half-duplex and Full-duplex

- The following shows how two devices can communicate with each other.

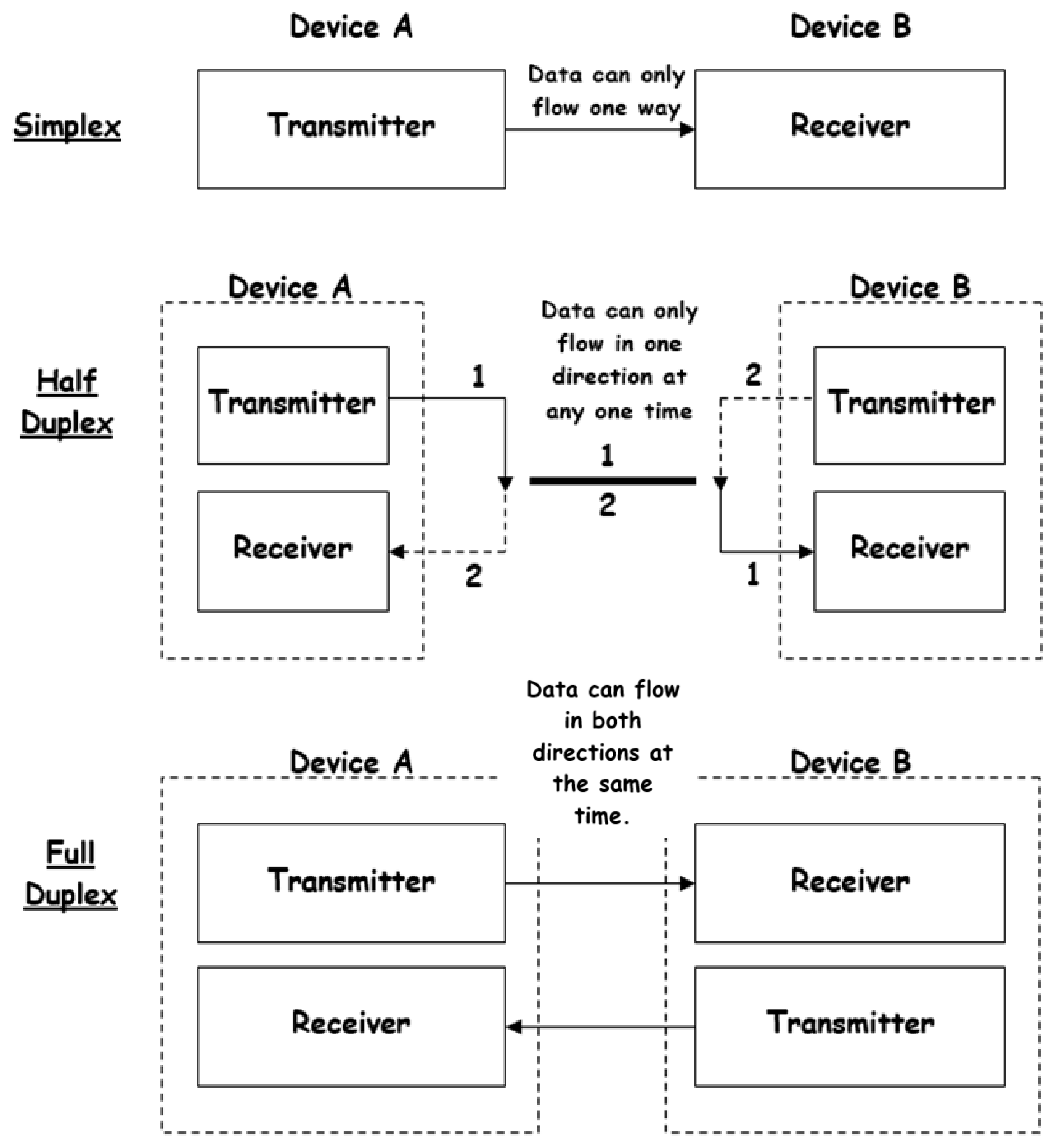


Figure 8.3 Simplex, half- and full-duplex transfers

8.1 Basics of serial communication

Serial communication protocol

- Before any 2 devices can communicate with one another, a communication protocol (i.e., a set of rules to make sense of the serial data) must first be defined to standardize how the data is packed, how many bits constitute a character, and when the data begins and ends.

- For instance, the ASCII 'A' or 0x41 or 0b01000001 can be framed in the following way shown in Figure 8.4 below.

8.1 Basics of serial communication

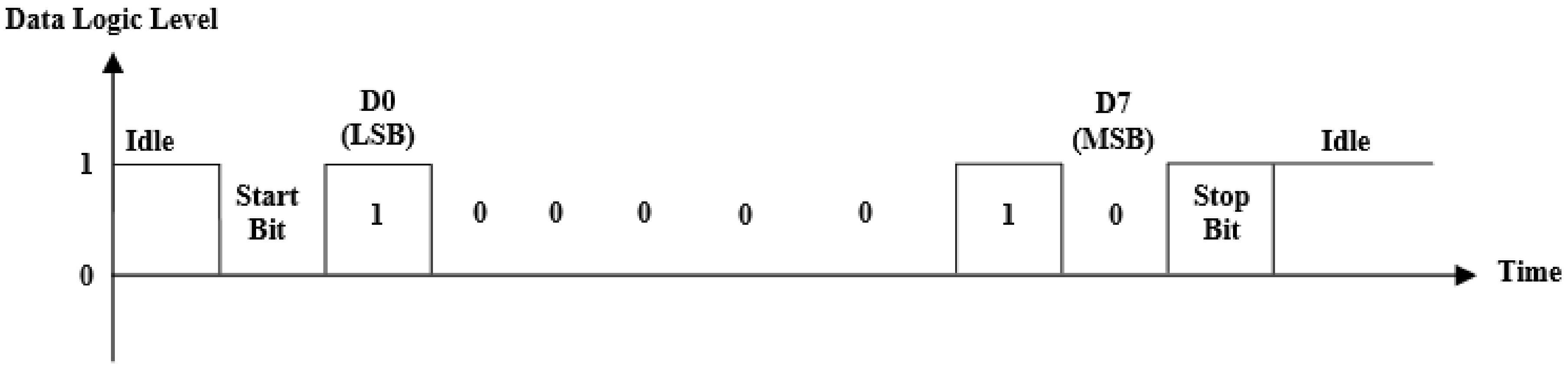


Figure 8.4 Framing ASCII 'A' with one start bit and one stop bit

8.1 Basics of serial communication

- The start bit is always one bit, but the stop bit can be one or two bits. The start bit is always a '0' (low) and the stop bit(s) is '1' (high).
- Note that LSB (D0) is sent out first.
- The data can be 7 bits wide or 8 bits wide (as in the above example).
- In some systems, the parity bit of the character byte is included in the data frame (before the stop bit) to maintain data integrity.
- The parity bit is odd or even. In the case of odd parity bit, the number of 1's in the data bit, including the parity bit, is odd.

8.1 Basics of serial communication

- However, it's essential to understand the technical differences between the two concepts. Baud rate refers to the number of signal changes or symbols transmitted per second, i.e.,

Baud rate = number of signal elements/ total time (in seconds)

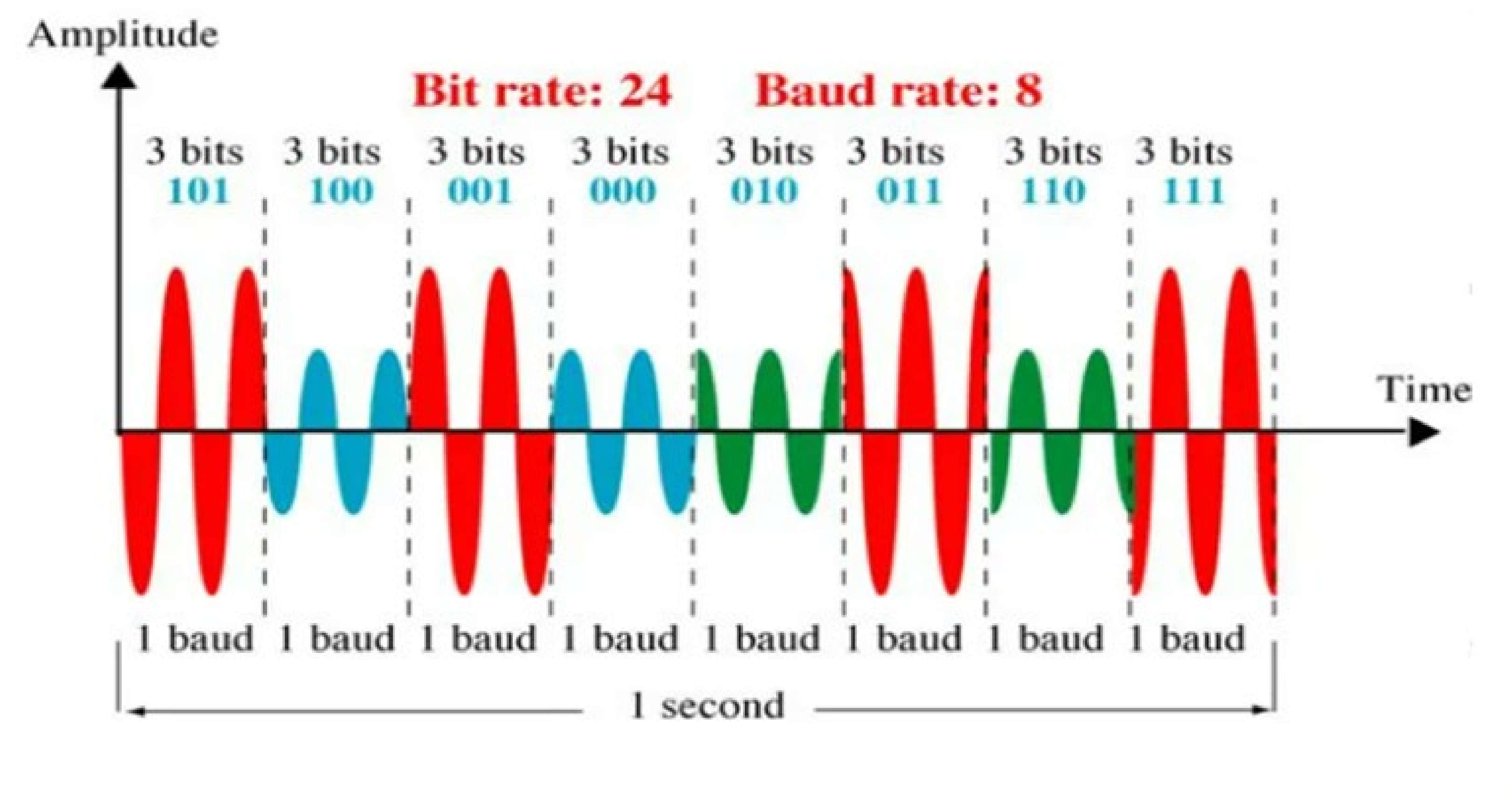
while bit rate refers to the number of bits transmitted per second, i.e.,

Bit rate = number of bits transmitted/ total time (in seconds)

So, Bit rate = Baud rate × Bits per symbol

8.1 Basics of serial communication

- Below shows a simple illustration of the difference between Baud rate and Bit rate (with 8-QAM: quadrature amplitude modulation).



8.1 Basics of serial communication

RS232 interfacing standards

- RS232 is the most widely used serial I/O interfacing standard, which allows PCs and numerous types of equipment made by different manufacturers to be connected to one another.
- In RS232, a ‘1’ is represented by -3 to -25 V while a ‘0’ is represented by +3 to +25 V.
- To connect any RS232 equipment to a microcontroller that produces TTL voltages (0 V for ‘0’ and 3.3V for ‘1’ or 5 V for ‘1’), a voltage converter such as MAX3232 (for 3.3V or 5V) or MAX232 (for 5V) can be used.

8.1 Basics of serial communication

- The connector used for the serial data cable can be male / female, 9-pin (so called DB9) or 25 pins (DB25).

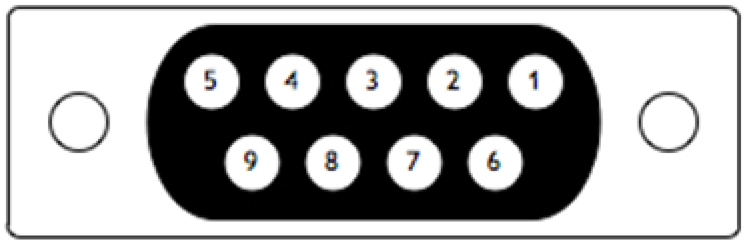


Figure 8.5 DB9 connector

8.1 Basics of serial communication

- RS232 classifies data communication equipment as DTE (data terminal equipment) or DCE (data communication equipment). DTE refers to terminals and computers that send and receive data, while DCE refers to communication equipment, such as modems for transferring the data.
- RS232 pin function definitions are shown in below table.

Table 8.1 RS232 pin function definitions

DB9 Pin	DB25 Pin	Acronym	Full name	Direction (DTE)	Meaning
1	8	DCD	Data Carrier Detect	DTE <-- DCE	Modem connected to another
2	3	RxD	Receive Data	DTE <-- DCE	Receives bytes into PC (DCE to DTE)
3	2	TxD	Transmit Data	DTE --> DCE	Transmits bytes out of PC (DTE to DCE)
4	20	DTR	Data Terminal Ready	DTE --> DCE	I'm ready to communicate (DTE)
5	7	SG	Signal Ground		
6	6	DSR	Data Set Ready	DTE <-- DCE	I'm ready to communicate (DCE)
7	4	RTS	Request To Send	DTE --> DCE	RTS/CTS flow control (DTE to DCE)
8	5	CTS	Clear To Send	DTE <-- DCE	RTS/CTS flow control (DCE to DTE)
9	22	RI	Ring Indicator	DTE <-- DCE	Telephone line ringing

8.1 Basics of serial communication

- The simplest connection between a PC and a microcontroller requires a minimum of 3 pins: Tx (Transmit), Rx (Receive) and ground, as shown below. Ensure that the Tx of one equipment goes to the Rx of the other equipment. Sometimes, other pins e.g., CTS (Clear To Send) are also used for "hand-shaking".

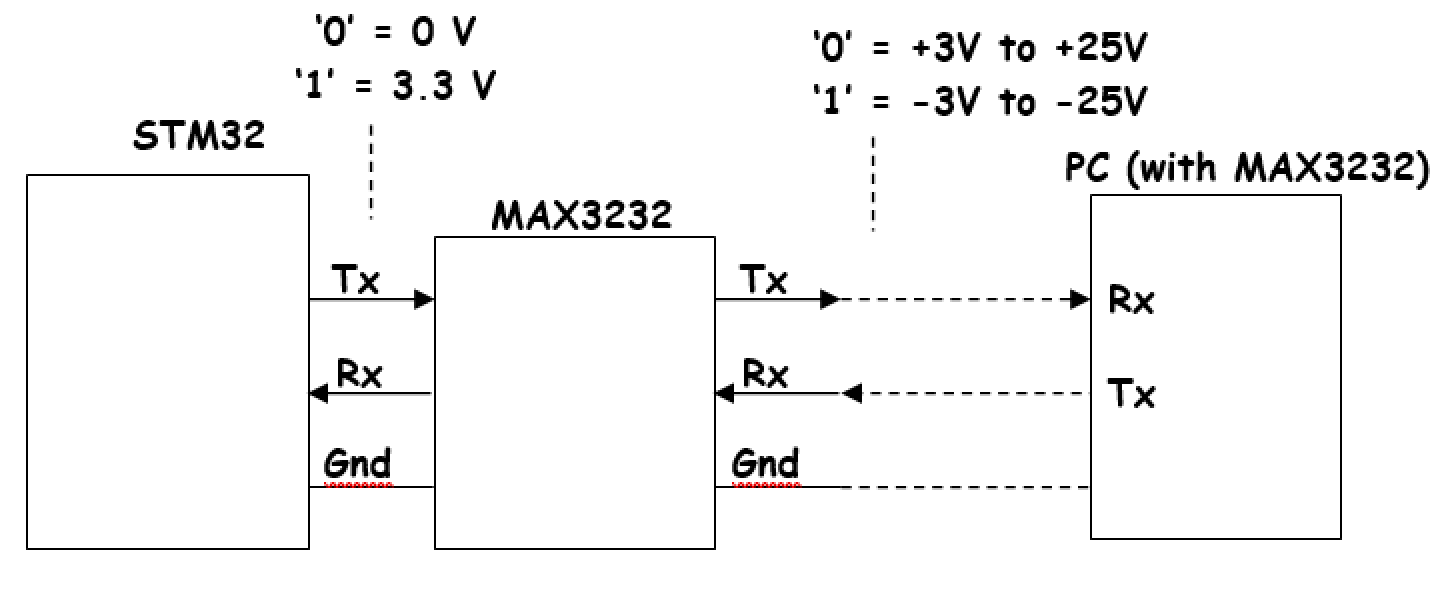


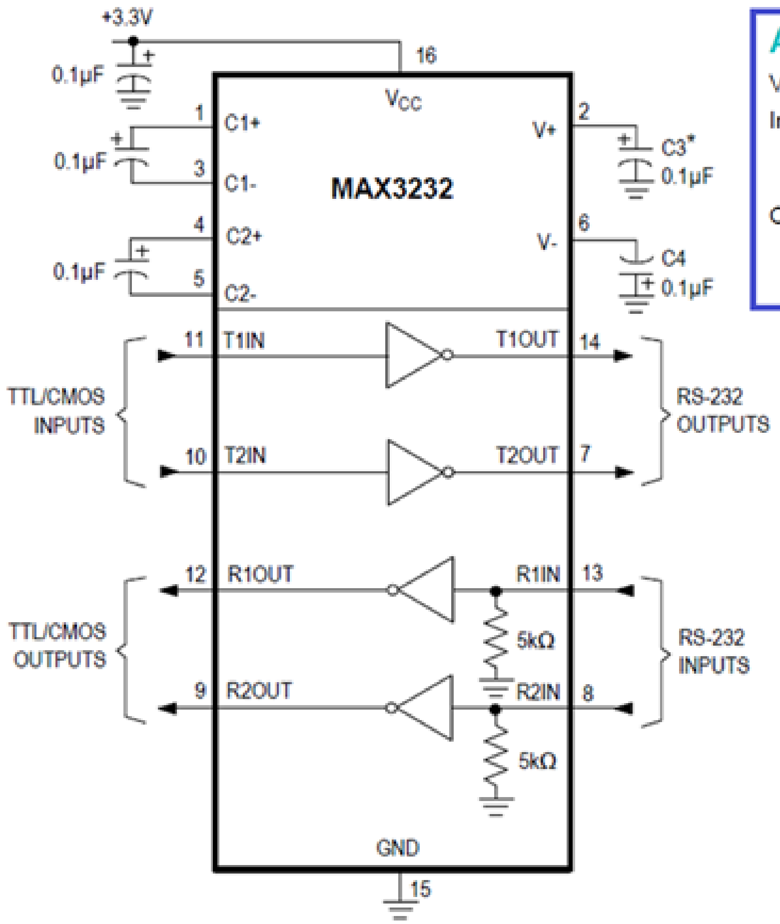
Figure 8.6 Microcontroller serial port to PC COM port

8.1 Basics of serial communication

- A STM32 MCU communicating with another STM32 MCU may communicate directly using TTL voltages (3.3V) with Tx and Rx be interchanged.
- Nowadays, COM ports (RS232 ports on a PC) are disappearing and replaced by USB ports. A "COM-to-USB converter" allows PC with only USB port to control devices with only RS232 interface e.g., a thermal label printer.

The MAX3232 device consists of two drivers and two receivers. It can provide the electrical interface between an asynchronous communication controller and the serial-port connector. A typical operating circuit for MAX3232 is shown below (MAX3222/MAX3232/MAX3237/MAX3241 - 3.0V to 5.5V, Low-Power, up to 1Mbps, True RS-232 Transceivers - analog.com)

8.1 Basics of serial communication



Absolute Maximum Ratings

V_{CC}	-0.3V to +6V
Input Voltages	
T_IN	-0.3V to +6V
R_IN	±25V
Output Voltages	
T_OUT	±13.2V
R_OUT	-0.3V to (V_{CC} + 0.3V)

Figure 8.7 typical operating circuit for MAX3232

8.2 STM32F103RB connection to RS232

- STM32F103RB has three built-in USARTs, as shown in below table (user can remap the alternate function to other pins if the original pins have been assigned for other functions). Notice that on the MAPP experiment board, only PA2 (USART2 Tx) and PA3 (USART Rx) pins are available for UART experiment. Other pins have been assigned for other purposes.

8.2 STM32F103RB connection to RS232

Table 8.2 STM32F103 USART pins

GPIO Pins	Alternate Functions	Remap
PA2	USART2_TX	
PA3	USART2_RX	
PA9	USART1_TX	
PA10	USART1_RX	
PB6		USART1_TX
PB7		USART1_RX
PB10	USART3_TX	
PB11	USART3_RX	
PC10		USART3_TX
PC11		USART3_RX

8.2 STM32F103RB connection to RS232

- The following diagram shows how a STM32F103RB can be connected to a MAX3232 (voltage converter) and then to a DB9 connector.

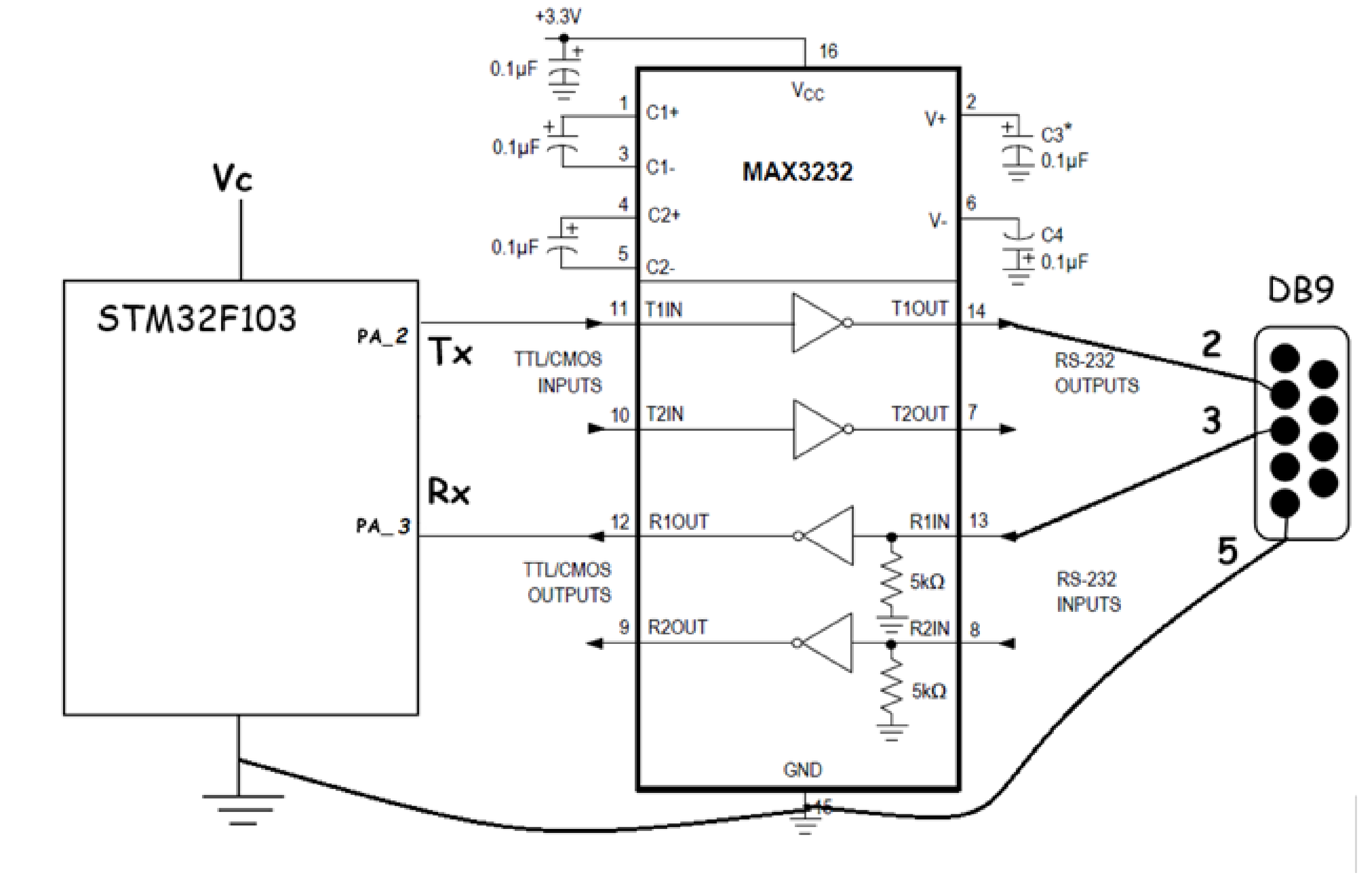


Figure 8.8 Connecting STM32F103RB to MAX3233 and then to DB9

8.3 STM32F103RB USART

- STM32F103RB has 3 USARTs, designated as USARTx, x = 1, 2, and 3. The base address for each USART is:
  - USART1: 0x4001 3800
  - USART2: 0x4000 4400
  - USART3: 0x4000 4800

8.3 STM32F103RB USART

- Below shows a simplified block diagram of USART.

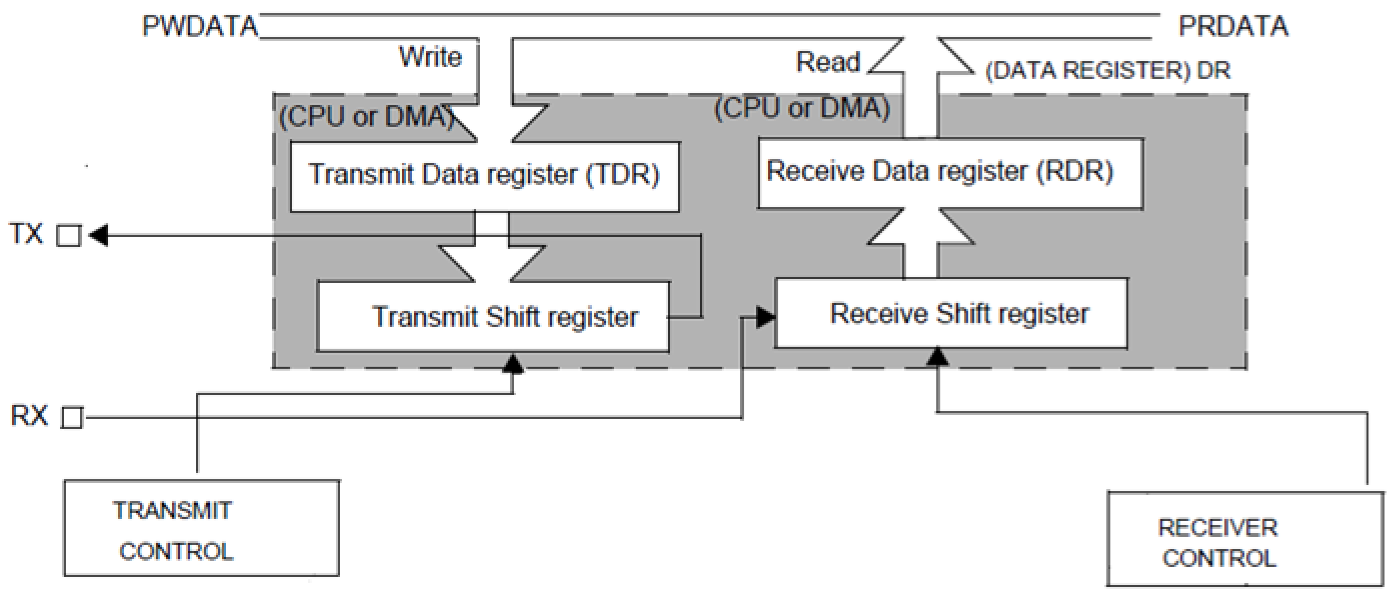


Figure 8.9 A simplified block diagram of USART

8.3 STM32F103RB USART

- Below table lists the relevant registers for each USART (STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced Arm®-based 32-bit MCUs - Reference manual).

Table 8.3 STM32F103 USART Registers

Register Name	Register Function	Register Address Offset (Hex)
USART_SR	Status Register	0x00
USART_DR	Data Register	0x04
USART_BRR	Baud Rate Register	0x08
USART_CR1	Control Register 1	0x0C
USART_CR2	Control Register 2	0x10
USART_CR3	Control Register 3	0x14
USART_GTPR	Guard Time and Prescaler Register	0x18

8.3 STM32F103RB USART

- For a more detailed USART registers' description, please refer to *8.7 Appendix I: USART registers.*
- Those registers can be categorized into 3 groups:
  - **Configuration registers:** Before using the USART peripheral the configuration registers must be initialized. This sets some parameters of the communication including Baud rate, word length, stop bit, interrupts (if needed). The configuration registers are: BRR, CR1, CR2, and CR3.

8.3 STM32F103RB USART

- **Transmit and receive register**: The DR register performs a double function (read and write) since it is composed of two registers, one for transmission (TDR) and one for reception (RDR). The TDR register provides the parallel interface between the internal bus and the output shift register. The RDR register provides the parallel interface between the input shift register and the internal bus. To send data, we simply write to the DR. To receive data, we simply read from the DR, if the received data has been stored in the RDR.
- **Interrupt and status register**: The SR register contains some flags to show the state of sending and receiving data including the availability of the newly received data, the existence of errors in the receiving data, the sending unit is ready for new data, interrupt flag, etc.

8.3 STM32F103RB USART

Setting up the Baud Rate

- We have learnt in STM32F103, there are different clock sources (e.g., HSI, HSE, PLL, etc.) to drive the system clock (SYSCLK). For NUCLEO F103RB board, the clock is set to use HSE (High Speed External) clock (8MHz) from the ST-Link as the PLL clock input.
- The SYSCLK is running at 72MHz, as shown below.

8.3 STM32F103RB USART

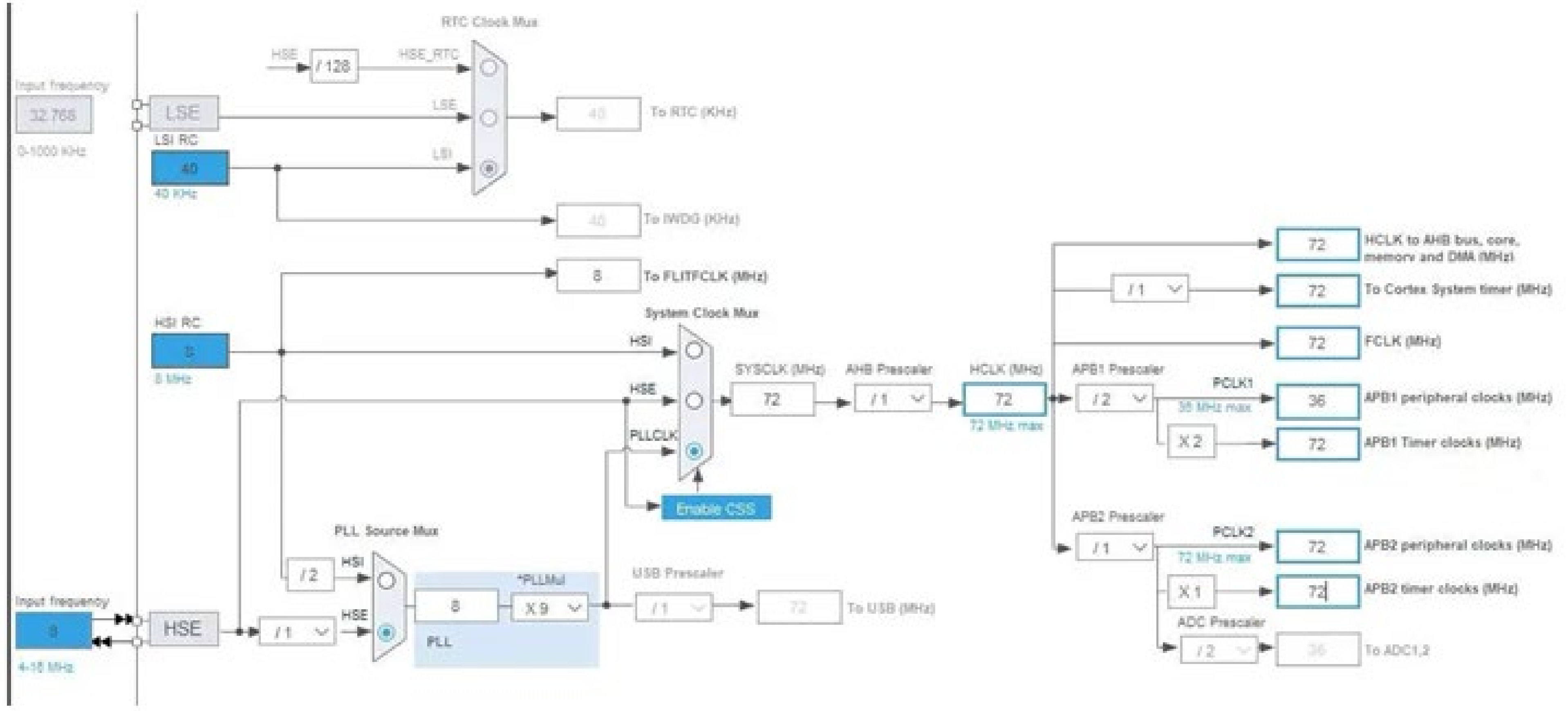


Figure 8.10 SYSCLK settings

8.3 STM32F103RB USART

- In STM32F103, USART1 is connected to the APB2 (Advanced Peripheral Bus 2) bus, however, USART2 and USART3 are connected to the APB1 bus.
- The USARTx clock enable bits are in the APB1ENR (APB1 enable register) and APB2ENR, as shown below.

APB1 peripheral clock enable register (RCC_APB1ENR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		DAC EN	PWR EN	BKP EN	CAN2 EN	CAN1 EN	Reserved		I2C2 EN	I2C1 EN	UART5EN	UART4EN	USART3 EN	USART2 EN	Res.
		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
SPI3 EN	SPI2 EN	Reserved		WWD GEN	Reserved					TIM7 EN	TIM6 EN	TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN
rw	rw			rw						rw	rw	rw	rw	rw	rw

Bit 18 **USART3EN**: USART 3 clock enable
Set and cleared by software.
0: USART 3 clock disabled
1: USART 3 clock enabled

Bit 17 **USART2EN**: USART 2 clock enable
Set and cleared by software.
0: USART 2 clock disabled
1: USART 2 clock enabled

8.3 STM32F103RB USART

APB2 peripheral clock enable register (RCC_APB2ENR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	USART 1EN	Res.	SPI1 EN	TIM1 EN	ADC2 EN	ADC1 EN	Reserved		IOPE EN	IOPD EN	IOPC EN	IOPB EN	IOPA EN	Res.	AFIO EN
	rw		rw	rw	rw	rw			rw	rw	rw	rw	rw		rw

Bit 14 **USART1EN**: USART1 clock enable

Set and cleared by software.

0: USART1 clock disabled

1: USART1 clock enabled

8.3 STM32F103RB USART

- In this chapter, we will use USART2 (Tx: PA2; Rx: PA3) to demonstrate how to program UART communications.
- From Figure 8.10, we can see APB1 Prescale is set as 2 to generate 36MHz (72MHz / 2) clock for APB1 peripheral clocks. Thus, USART2 clock is 36MHz.
- In a UART communication, the transmitter uses the transmission clock to pace the data bit output. For each clock pulse, one bit is transmitted. Therefore, the transmitter operates on the clock that runs at the Baud rate.

8.3 STM32F103RB USART

- Because UART is Asynchronous, the receiver needs to detect the falling edge of the start bit and then samples the consequent bits at the centre of the bit time, so it must run on a faster clock, which is called Oversampling. STM32F103 MCU implements the oversampling rate of 16 (i.e., each bit is sampled 16 times)
- The Baud rate for the receiver and transmitter (Rx and Tx) are both set to the same value as programmed in the USART_BRR register. Notice that although the USART_BRR register is a 32-bit register, only the lower 16 bits are used.
- The USART_BRR register is shown as below:

8.3 STM32F103RB USART

Baud rate register (USART_BRR)

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

Bits 31:16 Reserved, forced by hardware to 0.

Bits 15:4 **DIV_Mantissa[11:0]**: mantissa of USARTDIV
These 12 bits define the mantissa of the USART Divider (USARTDIV)

Bits 3:0 **DIV_Fraction[3:0]**: fraction of USARTDIV
These 4 bits define the fraction of the USART Divider (USARTDIV)

8.3 STM32F103RB USART

- The Baud rate equation is given as below:

$$\text{Tx/ Rx baud} = \frac{f_{CK}}{(16*\text{USARTDIV})}$$

legend: f_{CK} - Input clock to the peripheral (PCLK1 for USART2, 3, or PCLK2 for USART1)

- f_{CK} here for USART2 is 36MHz for NEUCLEO F103RB board.
- USARTDIV is an unsigned fixed-point number that is coded on the USART_BRR register. The baud counters are updated with the new value of the Baud registers after a write to USART_BRR. Hence the Baud rate register value should not be changed during communication.

8.3 STM32F103RB USART

- How to derive USARTDIV from USART_BRR register values
  - **Example 1:**

    If DIV_Mantissa = 27 and DIV_Fraction = 12 (*USART_BRR = 0x1BC*), then:

    Mantissa (USARTDIV) = 27

    Fraction (USARTDIV) = 12/16 = 0.75

    Therefore USARTDIV = 27.75

8.3 STM32F103RB USART

- **Example 2:**

To program USARTDIV = 25.62

This leads to:

DIV_Fraction = 16*0.62 = 9.92

The nearest real number is 10 = 0xA

DIV_Mantissa = mantissa (25.620) = 25 = 0x19

Then, USART_BRR = 0x19A hence USARTDIV = 25.625

8.3 STM32F103RB USART

- **Example 3:**

To program USARTDIV = 50.99

This leads to:

DIV_Fraction = 16*0.99 = 15.84

The nearest real number is 16 = 0x10 => overflow of DIV_frac[3:0] => carry must be added up to the mantissa:

DIV_Mantissa = mantissa (50.990 + carry) = 51 = 0x33

Then, USART_BRR = 0x330 hence USARTDIV = 51.000

8.3 STM32F103RB USART

- **Example 4:**

Find the values for the USART_BRR register for Baud rate of 9600bps:

USARTDIV = 36MHz / (16 x 9600) = 234.375

To program USARTDIV = 234.375

This leads to:

DIV_Fraction = 16*0.375 = 6 = 0x6

DIV_Mantissa = mantissa (234.375) = 234 = 0xEA

Then, USART_BRR = 0xEA6, when USARTDIV = 234.375

8.3 STM32F103RB USART

- Table 8.4 below shows the calculation for programmed Baud rates. Please calculate and verify for the Baud rate of 19200bps and 115200bps.

Baud rate		f_{PCLK} = 36 MHz		
S.No	in Kbps	Actual	Value programmed in the Baud Rate register	% Error[1]
1.	2.4	2.400	937.5	0%
2.	9.6	9.600	234.375	0%
3.	19.2	19.2	117.1875	0%
4.	57.6	57.6	39.0625	0%
5.	115.2	115.384	19.5	0.15%
6.	230.4	230.769	9.75	0.16%
7.	460.8	461.538	4.875	0.16%
8.	921.6	923.076	2.4375	0.16%
9.	2250	2250	1	0%
10.	4500	NA	NA	NA

1. Defined as (Calculated Baud Rate - Desired Baud Rate) / Desired Baud Rate.

Table 8.4 Calculation for programmed Baud rates

8.3 STM32F103RB USART

USART Control registers

- ☐ Among the USART Control registers, the most important are USART_CR1 and USART_CR2, which are both 32-bit registers.
- ☐ The USART_CR1 register is shown as below. It is used to set the number of bits per character (data length) in a frame with the default of 8 bits.

8.3 STM32F103RB USART

Control register 1 (USART_CR1)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved		UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 13 **UE**: USART enable

When this bit is cleared the USART prescalers and outputs are stopped and the end of the current
byte transfer in order to reduce power consumption. This bit is set and cleared by software.

0: USART prescaler and outputs disabled

1: USART enabled

Bit 12 **M**: Word length

This bit determines the word length. It is set or cleared by software.

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

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

Note: The M bit must not be modified during a data transfer (both transmission and reception)

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

Note: 1: During transmission, a "0" pulse on the TE bit ("0" followed by "1") sends a preamble (idle line) after the current word, except in Smartcard mode.

2: When TE is set there is a 1 bit-time delay before the transmission starts.

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

8.3 STM32F103RB USART

- It is also used to enable the serial port to send (TE, Transmit Enable) and receive (RE, Receive Enable) data too.
- Notice that USART feature cannot be used unless the UE (USART Enable) bit of USART_CR1 is set to 1.
- USART_CR2 register is shown below. It gives the selection of number of stop bits with the default of 1 stop bit.

Control register 2 (USART_CR2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	LINEN	STOP[1:0]		CLK EN	CPOL	CPHA	LBCL	Res.	LBDIE	LBDL	Res.	ADD[3:0]			
	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw

Bits 13:12 **STOP**: STOP bits

These bits are used for programming the stop bits.

00: 1 Stop bit

01: 0.5 Stop bit

10: 2 Stop bits

11: 1.5 Stop bit

8.3 STM32F103RB USART

USART Data register

- The USART_DR register is shown as below. Notice this is a 32-bit register and for 8-bit character size, only the lower 8bits are used.
- To transmit a byte of data, we must place it in USART_DR register. It must be noted that a Write to this register initiates a transmission from the USART.
- It must be also noted that received data in USART_DR register must be retrieved by reading it before it is lost.

8.3 STM32F103RB USART

- USART_DR register performs a double function (read and write) since it is composed of two registers, one for transmission (TDR) and one for reception (RDR). The TDR register provides the parallel interface between the internal bus and the output shift register. The RDR register provides the parallel interface between the input shift register and the internal bus.

8.3 STM32F103RB USART

Data register (USART_DR)

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

Bits 31:9 Reserved, forced by hardware to 0.

Bits 8:0 **DR[8:0]**: Data value

Contains the Received or Transmitted data character, depending on whether it is read from or written to.

The Data register performs a double function (read and write) since it is composed of two registers, one for transmission (TDR) and one for reception (RDR)

The TDR register provides the parallel interface between the internal bus and the output shift register (see Figure 1).

The RDR register provides the parallel interface between the input shift register and the internal bus.

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.

When receiving with the parity enabled, the value read in the MSB bit is the received parity bit.

8.3 STM32F103RB USART

USART Status register

Status register (USART_SR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NE	FE	PE
						rc_w0	rc_w0	r	rc_w0	rc_w0	r	r	r	r	r

Bit 7 **TXE**: Transmit data register empty

This bit is set by hardware when the content of the TDR register has been transferred into the shift register. An interrupt is generated if the TXEIE bit =1 in the USART_CR1 register. It is cleared by a write to the USART_DR 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 a software sequence (a read from the USART_SR register followed by a write to the USART_DR register). The TC bit can also be cleared by writing a '0' to it. This clearing sequence is recommended only for multibuffer communication.

0: Transmission is not complete

1: Transmission is complete

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_DR register. An interrupt is generated if RXNEIE=1 in the USART_CR1 register. It is cleared by a read to the USART_DR register. The RXNE flag can also be cleared by writing a zero to it. This clearing sequence is recommended only for multibuffer communication.

0: Data is not received

1: Received data is ready to be read.

8.3 STM32F103RB USART

- The USART_SR register is shown as above. Three of its bits are used by the USART extensively. They are:
  - TXE (Transmit Data Register Empty) Flag: this flag indicates that the transmit data register is empty and is available to take a new byte of data to be transmitted. We monitor the TXE flag before we write another byte to the USART_DR register.

8.3 STM32F103RB USART

- RXNE (Read Data Register Not Empty) Flag: this flag indicates data register has received a new byte of data and needs to be picked up before it is lost. The program should check the RXNE flag frequently to see if data is ready to be read. After the data is read from the USART_DR register, the RXNE flag is cleared automatically. If the program failed to read the data out of the USART_DR register before the next byte is shifted in, a buffer overflow error occurs. A byte of data is lost and the Overrun Error (ORE) flag in the USART_SR register is set.

8.3 STM32F103RB USART

- TC (Transmit Complete) Flag: this flag indicates that transmission is complete and there is no more data to be transmitted by USART, i.e., means it is ready to accept a new data to be transmitted.

8.4 Programming USART2

Reset and enabling the USART

- To conserve power, USART1, USART2 and USART3 are disabled coming out of reset. Before enabling the USART, first the bus clock to the USART should be enabled.
- The USART module should be configured properly before it is enabled because most of the control registers are enabled-protected, which means they cannot be modified while the module is enabled. When the configuration is done, writing a '1' to USART Enable bit (UE) of the USART_CR1 register enables the USART module.

8.4 Programming USART2

GPIO pins used for USART Tx and Rx

- In addition to the USART registers setup, we must also configure the GPIO pins for USART (Tx and Rx) to use their Alternate Functions.
- The GPIO pins in STM32F103 for USART are listed in Table 8.2. We use the PA2 (USART2 Tx) and PA3 (USART2 Rx) to demonstrate how to program the UART communications.
- To configure PA2 to use its Alternate Function, need to follow Table 3.2 and Table 3.3 to set the GPIOA_CRL register (shown as below).

8.4 Programming USART2

Port configuration register low (GPIOx_CRL) (x=A..G)

Address offset: 0x00

Reset value: 0x4444 4444

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNF7[1:0]		MODE7[1:0]		CNF6[1:0]		MODE6[1:0]		CNF5[1:0]		MODE5[1:0]		CNF4[1:0]		MODE4[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**
CNF3[1:0]		MODE3[1:0]		CNF2[1:0]		MODE2[1:0]		CNF1[1:0]		MODE1[1:0]		CNF0[1:0]		MODE0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:30, 27:26, 23:22, 19:18, 15:14, 11:10, 7:6, 3:2 **CNFy[1:0]:** Port x configuration bits (y= 0 .. 7)

These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table*.

In input mode (MODE[1:0]=00):

00: Analog mode
01: Floating input (reset state)
10: Input with pull-up / pull-down
11: Reserved

In output mode (MODE[1:0] > 00):

00: General purpose output push-pull
01: General purpose output Open-drain
10: Alternate function output Push-pull
11: Alternate function output Open-drain

Bits 29:28, 25:24, 21:20, 17:16, 13:12, 9:8, 5:4, 1:0 **MODEy[1:0]:** Port x mode bits (y= 0 .. 7)

These bits are written by software to configure the corresponding I/O port.
Refer to *Table 20: Port bit configuration table*.

00: Input mode (reset state)
01: Output mode, max speed 10 MHz.
10: Output mode, max speed 2 MHz.
11: Output mode, max speed 50 MHz.

8.4 Programming USART2

- PA_2 pin should be set as AF output, push-pull, at 10MHz in the GPIOA_CRL register. Sample code is shown below.

//configure PA_2 for UART2 TX

GPIOA->CRL &= 0xFFFFF0FF; //clear the CNF2[1:0] and MODE2[1:0]

GPIOA->CRL |= 0x0900; //set PA_2 as : AF output push-pull, 10MHz

8.4 Programming USART2

- For STM32F103, to use PA_3 pin as UART Rx, PA_3 pin should be set as Input, Pull-up/Pull-down in the GPIOA_CRL register and according to Table 3.2, the corresponding GPIOA_ODR (port output data register) register bit needs to be set as '1' to further configure PA_3 as Input Pull-up. Sample code is shown below.

```
//configure PA_3 for UART2 RX
GPIOA->CRL &= 0xFFFF0FFF; // clear the CNF3[1:0] and MODE3[1:0]
GPIOA->CRL |= 0x8000;    // set PA_3 as : Input, Pull up/down
GPIOA->ODR &= 0xFFFFFFF7;      //clear the bit ODR3 for PA_3
GPIOA->ODR |= 0x0008;      //set bit ODR3 to set PA_3: Input Pull-up
```

8.4 Programming USART2

Steps to configure USART2 for transmitting data

1) Enable the Clock to GPIOA.
2) Enable the Clock to USART2.
3) Configure PA_2 to use Alternate Function.
4) Set the Baud rate for USART2 using USART2_BRR register.
5) Configure the CR1 register for character size and enabling transmit (TE).
6) Configure the CR2 register for number of stop bit(s).
7) Configure the CR3 register for no hardware flow control.
8) Enable USART2 after configuration complete.
9) Wait until the TXE bit of the USART_SR register is set.
10) Write a byte to DR register to be transmitted.
11) To transfer the next character, go to step 9.

8.4 Programming USART2

Steps to configure USART2 for receiving data

1) Enable the Clock to GPIOA.
2) Enable the Clock to USART2.
3) Configure PA_3 as Input Pull-up pin (for STM32F103).
4) Set the Baud rate for USART2 using USART2_BRR register.
5) Configure the CR1 register for character size and enabling receive (RE).
6) Configure the CR2 register for number of stop bit(s).
7) Configure the CR3 register for no hardware flow control.
8) Enable USART2 after configuration complete.
9) Wait until the RXNE bit of the USART_SR register is set.
10) Read a byte from DR register that was received.
11) To receive the next character, go to step 9.

8.4 Programming USART2

Example: Echoing the received character from USART2

- Mbed Studio has a COM window (as shown below) which can communicate with the NUCLEO F103 Board through USART2. Baud rate is set 9600bps by default.

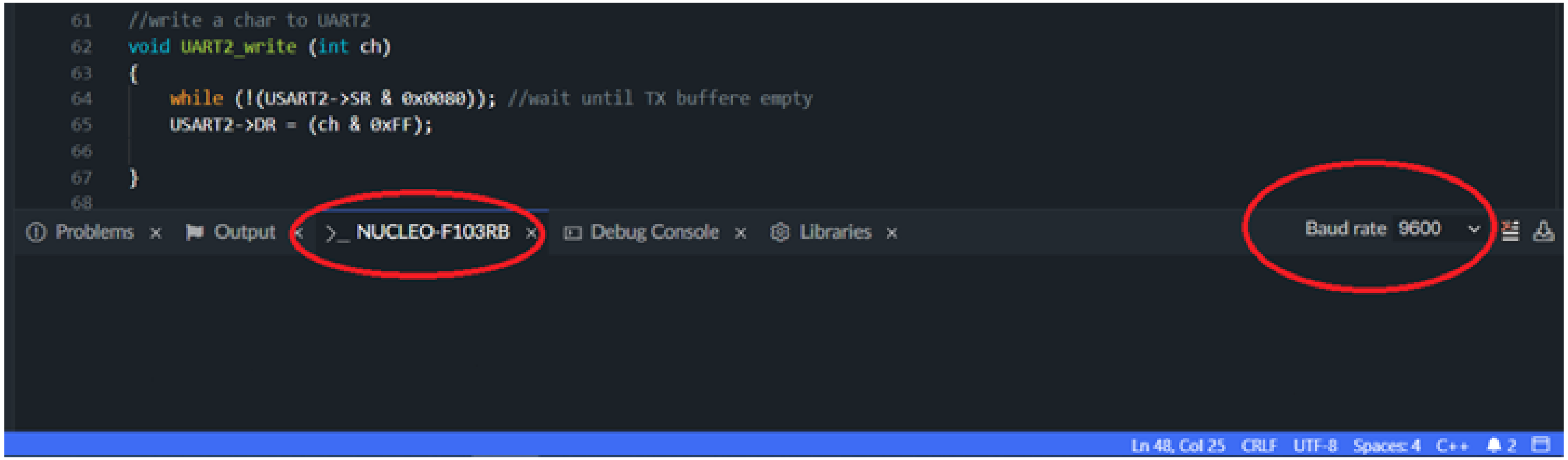


Figure 8.11 The COM window in Mbed Studio

8.4 Programming USART2

- Below sample code configures PA_2 (USART2 Tx) and PA_3 (USART2 Rx) so that a "Hello!" message can be sent to Mbed Studio COM window upon the program starts.
- After that, PA_3 waits for any key (character) being pressed in the Mbed Studio COM window. Once PA_3 receives a character, the program will "echo" back the character to the Mbed Studio COM window. Then we can see what we have keyed in and sent to the STM32F103 MCU.

8.4 Programming USART2

1

```
#undef __ARM_FP
#include "mbed.h"
#define UART2_TX    PA_2
#define UART2_RX    PA_3
void UART2_init (void);
void UART2_write (int ch);
char UART2_read (void);
unsigned char message_send[] = "Hello!\n";

int main(void)
{
    char chData;
    int i;

    UART2_init();
    for (i = 0; i < 7; i++)
    {
        UART2_write(message_send[i]);   //send
1 character by 1 character
    }
```

2

```
while (1)
    {
        chData = UART2_read();
        UART2_write(chData);
    }

 }
//Initialize UART2
void UART2_init (void)
{
    //enable the clocks
    RCC->APB1ENR |= 0x05;    //enable GPIOA clock and AFIO
clock
    RCC->APB1ENR |= 0x20000; //enable UART2 clock

    //configure PA_2 for UART2 TX
    GPIOA->CRL &= 0xFFFFF0FF;    //clear the CNF2[1:0] and
MODE2[1:0]
    GPIOA->CRL |= 0x0900;    //set PA_2 as: AF output push-
pull, 10MHz
```

1

```
//configure PA_3 for UART2 RX
   GPIOA->CRL &= 0xFFFF0FFF;      //clear the CNF3[1:0] and MODE3[1:0]
   GPIOA->CRL |= 0x8000;    //set PA_3 as: Input Pull-up/down
   GPIOA->ODR &= 0xFFFFFFF7;       //clear the bit ODR3 for PA_3
   GPIOA->ODR |= 0x0008;       //set bit ODR3 to set PA_3: Input Pull-up

  USART2->BRR = 0x0EA6;   //SYSCLK = 72MHz, APB1 = 36MHz,
                    //oversampling by 16 and Baud rate 9600bps
   USART2->CR1 = 0x000C;   //enable TX and RX
   USART2->CR2 = 0x0000;   //1 stop bit
   USART2->CR3 = 0x0000;      //no hardware flow control
   USART2->CR1 |= 0x2000; //enable USART
}
```

2

```
//write a char to UART2
void UART2_write (int ch)
{
   while (!(USART2->SR & 0x0080)); //wait until TX buffere empty
   USART2->DR = (ch & 0xFF);
}

//read a char from UART2
char UART2_read (void)
{
   while (!(USART2->SR & 0x0020)); //wait until char arrives
   return USART2->DR;
}
```

8.4 Programming USART2

☐ Below shows an example after the program runs.

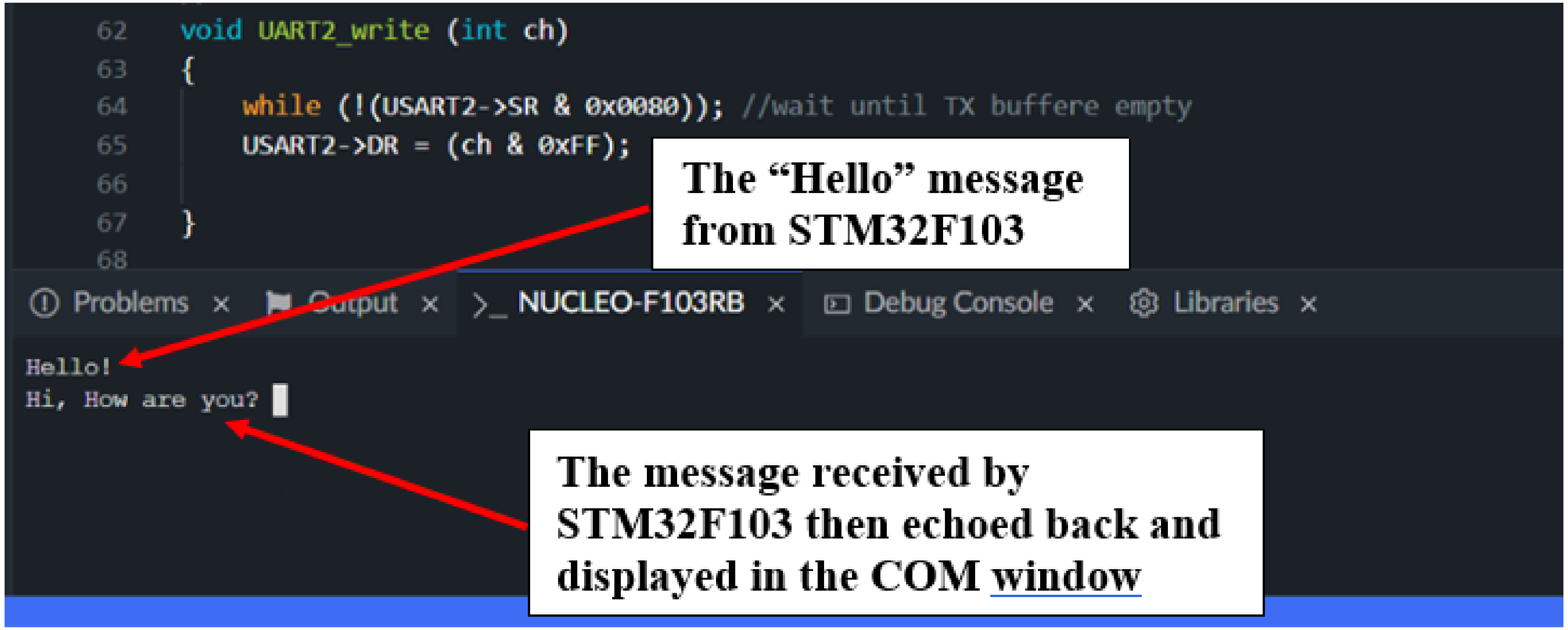


Figure 8.12 Example of the message "echo" program

8.5 Mbed Serial (UART) API - BufferedSerial

- Mbed Serial (UART) APIs provide a BufferedSerial class (BufferedSerial - API references and tutorials | Mbed OS 6 Documentation) to realize the UART functionalities.
- BufferedSerial class can help us quickly configure the USART and create a serial communication with other devices (such as sensors, printers or another board to exchange data or to send text to be displayed on a text-based computer interface.
- The BufferedSerial calls the underlying HAL API functions. When the receive interrupt is triggered when receiving data from a device, the BufferedSerial class stores the byte(s) available to read from the hardware buffer to an internal intermediary buffer.

8.5 Mbed Serial (UART) API - BufferedSerial

- To transmit multiple bytes, the class uses the intermediary buffer to store the bytes to send and monitors the serial interface to transfer them to the hardware buffer as soon as it is available.
- Below shows the BufferedSerial class reference.

8.5 M

BufferedSerial Class Reference

Public Member Functions	
	BufferedSerial (PinName tx, PinName rx, int baud=MBED_CONF_PLATFORM_DEFAULT_SERIAL_BAUD_RATE)
	Create a BufferedSerial port, connected to the specified transmit and receive pins, with a particular baud rate. More...
	BufferedSerial (const serial_pinmap_t &static_pinmap, int baud=MBED_CONF_PLATFORM_DEFAULT_SERIAL_BAUD_RATE)
	Create a BufferedSerial port, connected to the specified transmit and receive pins, with a particular baud rate. More...
short	poll (short events) const final
	Equivalent to POSIX poll(). More...
ssize_t	write (const void *buffer, size_t length) override
	Write the contents of a buffer to a file. More...
ssize_t	read (void *buffer, size_t length) override
	Read the contents of a file into a buffer. More...
int	close () override
	Close a file. More...
int	isatty () override
	Check if the file in an interactive terminal device. More...
off_t	seek (off_t offset, int whence) override
	Move the file position to a given offset from from a given location. More...
int	sync () override
	Flush any buffers associated with the file. More...
int	set_blocking (bool blocking) override

	Set blocking or non-blocking mode The default is blocking. More...
bool	is_blocking () const override
	Check current blocking or non-blocking mode for file operations. More...
int	enable_input (bool enabled) override
	Enable or disable input. More...
int	enable_output (bool enabled) override
	Enable or disable output. More...
void	sigio (Callback< void()> func) override
	Register a callback on state change of the file. More...
void	set_data_carrier_detect (PinName dcd_pin, bool active_high=false)
	Setup interrupt handler for DCD line. More...
void	set_baud (int baud)
	Set the baud rate. More...
void	set_format (int bits=8, Parity parity=BufferedSerial::None, int stop_bits=1)
	Set the transmission format used by the serial port. More...
void	set_flow_control (Flow type, PinName flow1=NC, PinName flow2=NC)
	Set the flow control type on the serial port. More...

8.5 Mbed Serial (UART) API - BufferedSerial

Example I: Using BufferedSerial to send message through USART2

- Below sample code uses BufferedSerial class to send a simple message of "This is a test!" to the Mbed Studio COM window repeatedly through USART2.
- It shows how to create an object of the BufferedSerial class, how to configure the UART and how to send out a multi-character message through USART2.
- After running the program, the Mbed Studio COM window keeps showing "This is a test!", as shown below.

8.5 Mbed Serial (UART) API - BufferedSerial

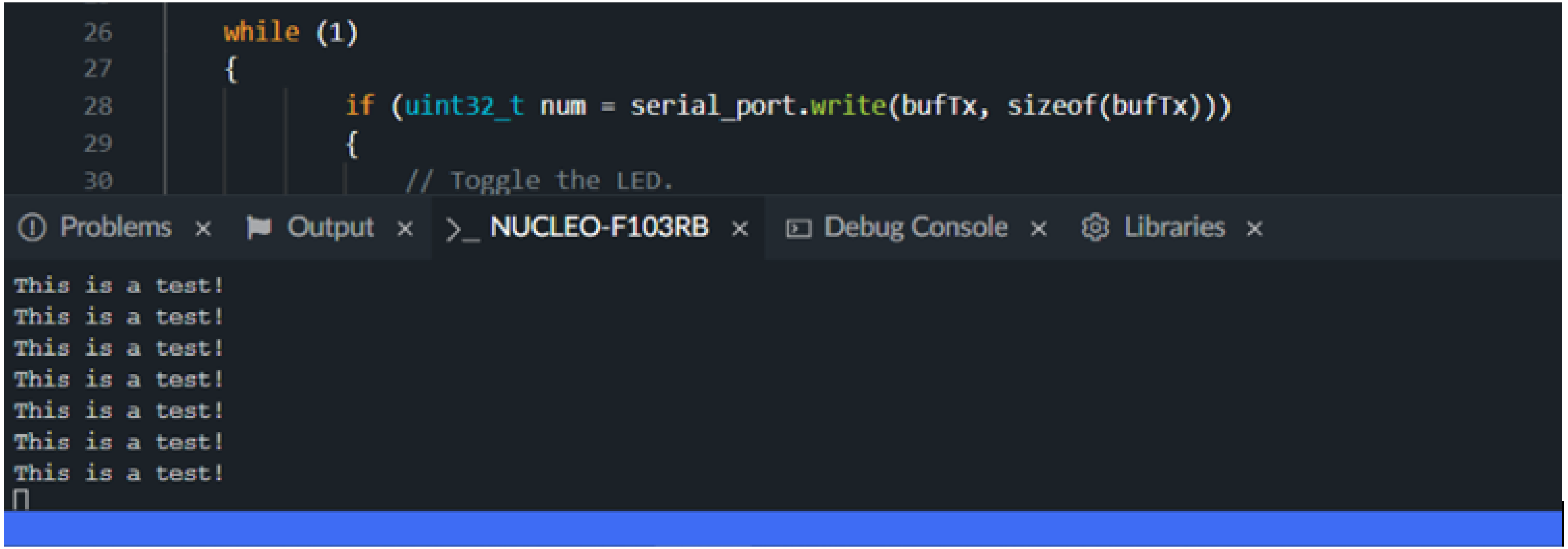


Figure 8.13 Example of sending a message using BufferedSerial class

8.5 Mbed Serial (UART) AP

- The sample code is shown below.

1

```
#undef __ARM_FP
#include "mbed.h"
// Create a DigitalOutput object to toggle an LED
static DigitalOut led(PB_14);

// Create a BufferedSerial object with a default
baud rate.
//PA_2: TX, PA_3: RX
static BufferedSerial serial_port(PA_2, PA_3);

int main(void)
{
   // Set desired properties (9600-8-N-1).
  serial_port.set_baud(9600); //set Baud rate
```

2

```
//set frame format
 serial_port.set_format(
     /* bits */ 8,
     /* parity */ BufferedSerial::None,
     /* stop bit */ 1
 );

 // Application buffer to send the data
 char bufTx[] = {"This is a test!\n"};

 while (1)
{
        //send the message through UART
       if (uint32_t num = serial_port.write(bufTx,
sizeof(bufTx)))
       {
          // Toggle the LED.
          led = !led;
          thread_sleep_for(1000);
       }
 }
}
```

8.5 Mbed Serial (UART) API - BufferedSerial

Example II: Using BufferedSerial to "echo" message

- From last example, we have learnt how to create the BufferedSerial class object and how to use the BufferedSerial class member functions, such as set_baud(), set_format(), and write() to send message to other device. In this example, we are going to learn how to use the read() member function to receive the message from other device.
- Below sample code uses BufferedSerial class to "echo" the received characters back to the Mbed Studio COM window.
- In addition, this sample code also displays the received characters on the LCD screen on the experiment board.

8.5 Mbed Serial (UART) API - BufferedSerial

- After the program is running, it sends out the message of "This is a test!" to the Mbed Studio COM window through USART2, then waits for receiving the characters from the Mbed Studio COM window.
- Below shows an example after keying some characters and displaying those characters being echoed by the STM32F103 MCU. As well, those characters are displayed on the LCD screen on the experiment board.

8.5 Mbed Serial (UART) API - BufferedSerial

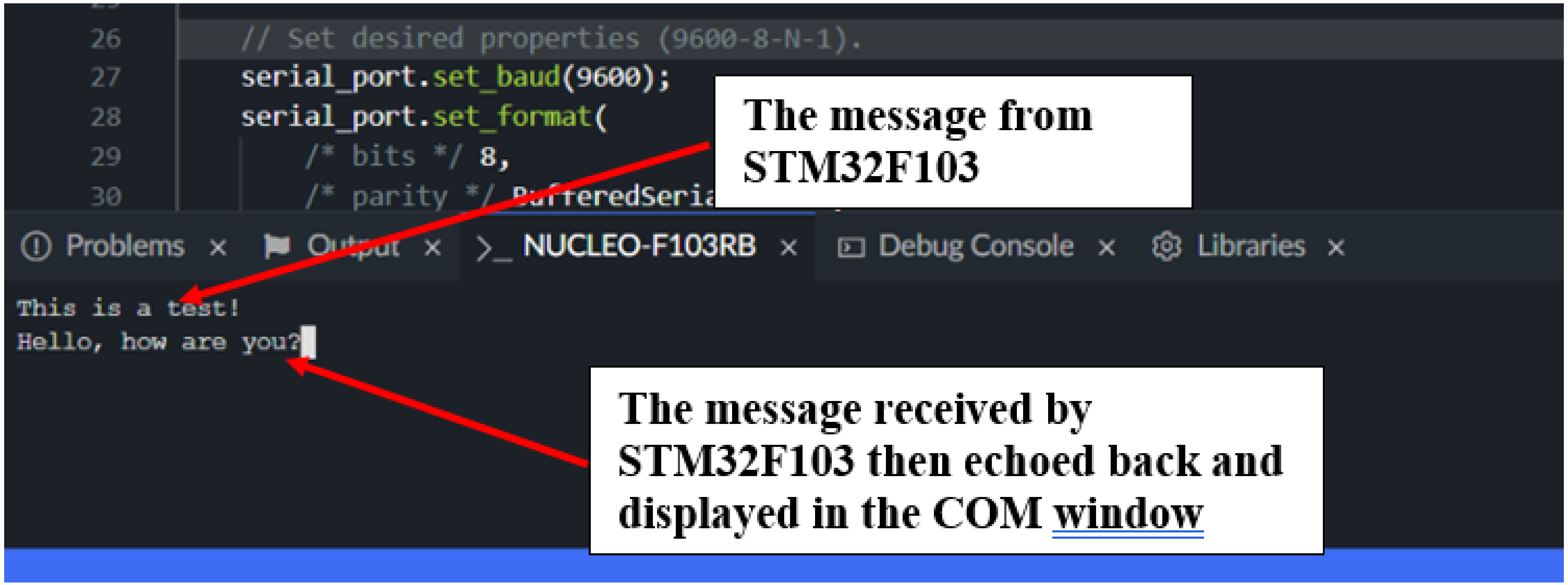


Figure 8.14 Receiving and echoing a message using BufferedSerial

8.5 Mbed Serial (UART) API - BufferedSerial

- You can key in more characters, such as “I am fine.” All the keyed in characters will be echoed and displayed in the COM window, as well as on the LCD screen on the experiment board, like what shown below.

8.5 Mbed Serial (UART) API - BufferedSerial

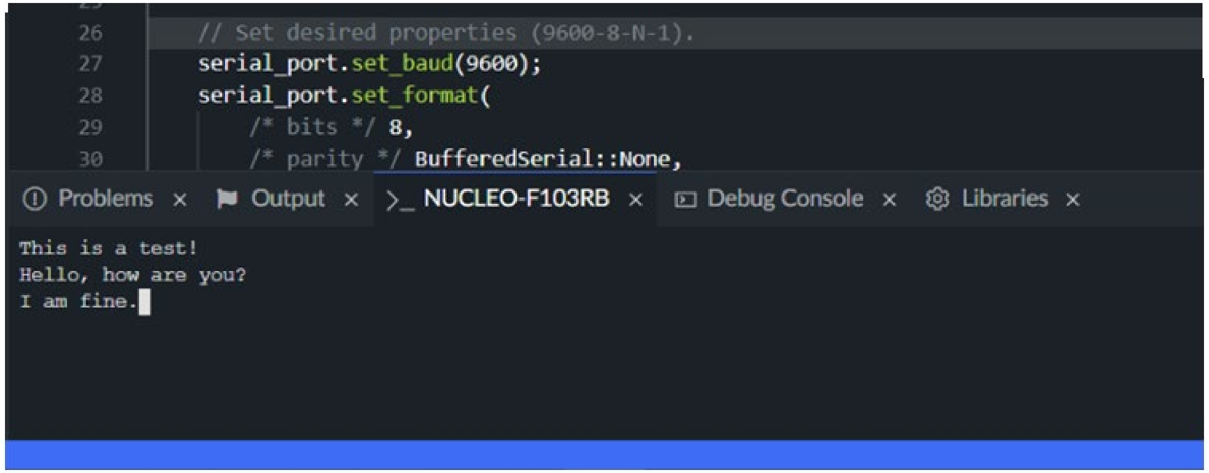


Figure 8.15 Message is displayed in COM window and on LCD.

8.5 Mbed Serial (UART) API - BufferedSerial

- The sample code is shown below. Notice here, the “Enter” key is displayed as a special character on the LCD. You may add in some codes to handle the “Enter” key without showing the special character.

8.5 Mbed Serial (UART) API

1

```
#undef __ARM_FP
#include "mbed.h"
#include "lcd.h"
// Maximum number of element the application buffer can contain
#define MAXIMUM_BUFFER_SIZE    32
#define UART2_TX    PA_2
#define UART2_RX    PA_3
// Create a DigitalOutput object to toggle an LED
//PB_14-> send data, PB_15-> receive data
static DigitalOut led_PB14(PB_14);
static DigitalOut led_PB15(PB_15);
// Create a BufferedSerial object with a default baud rate.
//PA_2: TX, PA_3: RX
static BufferedSerial serial_port(UART2_TX, UART2_RX);
```

2

```
int main(void)
{
   lcd_init();                                    // Initialise LCD module
   lcd_write_cmd(0x80);          // Move cursor to line 1 position 1

   // Set desired properties (9600-8-N-1).
   serial_port.set_baud(9600);
   serial_port.set_format(
      /* bits */ 8,
      /* parity */ BufferedSerial::None,
      /* stop bit */ 1
   );

   // Application buffer to send the data
   char bufRx[MAXIMUM_BUFFER_SIZE] = {0};
   char bufTx[] = {"This is a test!\n"};
  unsigned char nCount = 0;   //totoal number of char received and
                        //displayed on LCD, 1 line of LCD is
                    20 chars
   unsigned char nLine = 1;   //start with 1st line
```

3

```
if (uint32_t num1 = serial_port.write(bufTx, sizeof(bufTx)))
    {
        // Toggle the LED.
        led_PB14 = !led_PB14;
        thread_sleep_for(50);
    }
    while (1)
    {
        if (uint32_t num2 = serial_port.read(bufRx, sizeof(bufRx)))
        {
            // Toggle the LED.
            led_PB15 = !led_PB15;
            lcd_write_data(bufRx[0]);            // write to LCD
            nCount ++;
            if (nCount >= 20)
            {
                if (nLine == 1)
                {
                    lcd_write_cmd(0xC0); // Move cursor to line 2 position 1
                    nLine ++;
                    nCount = 0;
                }
```

4

```
                else
                {
                    lcd_Clear();
                    lcd_write_cmd(0x80);  // Move cursor to line 1 position 1
                    nLine = 1;
                    nCount = 0;
                }
            }

            // Echo the input back to the terminal.
            if (bufRx[num2-1] == 0x0A || bufRx[num2-1] == 0x0D)
            {
                bufRx[num2] = '\n';
                serial_port.write(bufRx, num2+1);
            }
            else
            {
                serial_port.write(bufRx, num2);
            }
            thread_sleep_for(50);
        }
    }//while
}//main
```

8.6 Review Questions

1. What do you think the receiver must have to convert the serial data it receives into 8-bit data that it usually handles?

2. If even parity is used, what is the parity bit for the ASCII 'A' i.e. 0b01000001?

3. Which member function in the Mbed Serial (UART) API: BufferedSerial Class allows one to set frame specifications such as number of data bits, parity bit, and stop bit?