Bern University
of Applied Sciences

Exercise - 09

Universal Synchronous and Asynchronous Receiver-Transmitter

Andreas Habegger | Adrian Steiner
BTS3230 | Version 1.0.0 of 17.03.2025

Please be aware that the content is subject to change at any time. For the latest version, please check the website.

In this exercise, you will implement communication via **U(S)ART** (Universal (Synchronous) and Asynchronous Receiver-Transmitter). The nucleo board acts as a UART peripheral, and your computer acts as the UART master (virtual serial interface over ST-Link). By sending the characters **r**, **g** and **b** to the nucleo board, you can control the color of the RGB LED.

Objectives

- U(S)ART - Universal (Synchronous) and Asynchronous Receiver-Transmitter
- Receive messages from computer
- Sending messages to the computer
- Use custom communication protocol

Outcomes

- Configure UART with desired baud rate
- Receive messages from your computer
- Send an echo from the MCU back to your computer
- Implement your own message protocol to interpret commands
- Control the LED with your computer

Description

UART is a common standard to interface another device from a computer. Since a computer nowadays does not have a direct UART interface, ST-Link over USB is used as a **Virtual COM Port**. In this case, communication between the STM32F446re and the ST-Link is established via UART. The ST-Link itself forwards the UART over USB as a virtual COM port. To control the LED, only a unidirectional communication from the computer to the nucleo board would be needed. To check that the hole communication is working, the bi-directional communication is used to send an echo back to the sender device. The nucleo board will poll the UART peripheral connected to the ST-Link virtual COM port for new messages and immediately send back the same message. A valid message triggers an action in your MCU firmware. Valid message symbols are the characters **r**, **R**, **g**, **G** and **b**, **B** for toggling the corresponding color channel of the RGB LED.

Tasks

Terminal

You will need a serial terminal emulator to send a message via UART from your computer to the nucleo board.

> **Note**
>
> This is not the same as a command terminal. However, you can use commands on a command terminal to receive and send data over the UART, but it is much more convenient for beginners to use a special program to do this.

There are several programs available to help with this. However, we recommend the Visual Studio extension VS-Code Serial Monitor or the standalone program HTerm. Serial Monitor can be installed in the vs code explorer (open with `^ Ctrl` + `P`) with:

```
ext install ms-vscode.vscode-serial-monitor
```

One standalone tool mentioned for Windows and GNU/Linux is HTerm. Terminal-based emulators for GNU/Linux can also be used like screen, minicom, or putty. Feel free to use whichever you like.

UART

For this exercise, we will use **USART2** for UART communication. Therefore, initialize USART2 to receive messages at a baud rate of $9600\,\mathrm{Bd}$. USART2 is used because the Rx and Tx pins of this peripheral are connected to the ST-Link UART Rx and Tx interface. The initialization is described in detail in the reference manual.

See also

Refer to section 25.4.2 from RM0390

Within the superloop, poll the RX register of USART2 for new characters. If a message is received, immediately return it to the sender as an echo. In addition, if a valid command is received, toggle the associated LED. Any other invalid message must be ignored.

Note

UART commands for the RGB LED control:

- **r** | **R** – Switches the red color channel
- **g** | **G** – Switches the green color channel
- **b** | **B** – Switches the blue color channel

USART2 Configuration:

- Configure alternate function for U(S)ART pins. Please refer to the schematic and the DS10693 table 11.
- Set the **Baud Rate**: $9600\,\mathrm{Bd}$.
- **Frame Properties**: 8-N-1

LED

Initialize and use the LED as previous exercises. If you receive a correct message, the corresponding LED will be toggled. Use the techniques you have learnt previously.

Implementation

Your implementation is based on three steps. At the end, there are additional options to extend your implementation.

LED

1. Initialise the LED as you have done many times before.
2. Implement a useful function to toggle the LED.

Next, implement the UART peripheral to read the messages with polling in the superloop and return the message as an echo to the sender.

USART2 echo

3. Configure the used communication GPIOs with the correct AF from the DS10693 table 11.
4. Initialise UART with the configuration defined in the UART section.

Initialisation UART

```
void usart_init(USART_TypeDef *uartHandler) {
  // Disable USART
  CLEAR_BIT(uartHandler->CR1, USART_CR1_UE);
  // Set data length to 8
  CLEAR_BIT(uartHandler->CR1, USART_CR1_M);
  // Select 1 stop b i t
  CLEAR_BIT(uartHandler->CR2, USART_CR2_STOP_0);
  CLEAR_BIT(uartHandler->CR2, USART_CR2_STOP_1);
  // Set parity control to no parity
  CLEAR_BIT(uartHandler->CR1, USART_CR1_PCE);
  // Set oversampling to 16
  CLEAR_BIT(uartHandler->CR1, USART_CR1_OVER8);
  // Set Baud Rate 115200 on a 16 MHz system
  // WRITE_REG(uartHandler- > BRR, 0x8B);
  // Set Baud Rate 115200 on a 8 MHz system
  // WRITE_REG(uartHandler- > BRR, 0x45);
  // Set Baud Rate 9600 on a 16MHz system
  // WRITE_REG(uartHandler->BRR, 0x683);
  // Set Baud Rate 9600 on a 8MHz system
  // WRITE_REG(uartHandler- > BRR, 0x683);
  // Enable transmission and receiving
  SET_BIT(uartHandler->CR1, (USART_CR1_TE | USART_CR1_RE));
  // Enable USART
  SET_BIT(uartHandler->CR1, USART_CR1_UE);
}
```

5. Implement a blocking one byte USART read function

Example blocking uart read function

```
void usart_blockingRead(USART_TypeDef *uartHandler, uint8_t *buffer) {
  if (NULL == uartHandler || NULL == buffer) {
    return;
  }

  // Wait until new data received in the DR
  while (!(READ_BIT(uartHandler->SR, USART_SR_RXNE))) {
    asm("NOP");
  }
  *buffer = (uint8_t)READ_REG(uartHandler->DR);
}
```

6. Implement a blocking UART write function

Example blocking uart write function

```
void usart_blockingWrite(USART_TypeDef *uartHandler,
    uint8_t *buffer,
    uint32_t bufferLength) {
  if (NULL == uartHandler || NULL == buffer || 0 == bufferLength) {
    return;
  }

  // Transmit multiple bytes
  for (uint32_t idx = 0; idx < bufferLength; idx++) {
    // Wait for the transfer from DR to tx shift register
    while (!(READ_BIT(uartHandler->SR, USART_SR_TXE))) {
      asm("NOP");
    }

    // Write out one byte.
    WRITE_REG(uartHandler->DR, buffer[idx]);
  }

  // Wait for transmission completion
  while (!(READ_BIT(uartHandler->SR, USART_SR_TC))) {
    asm("NOP");
  }
}
```

7. Implement an echo device that receives a message and immediately returns it to the sender.
8. Test your implementation by sending messages using your favourite serial terminal emulator on your computer.

Hint

Remember to use the same USART settings as on the MCU and use the correct USB port.

When you are satisfied that your USART communication is working, finish the exercise.

Finishing exercise

9. In order to recognise the desired commands, write a parser for the received message. Messages that do not match the rules must be ignored.
10. Depending on the command, switch the corresponding LED.

(optional) Redesign of communication protocol

The current implementation is not very user-friendly. Just getting an echo is not helpful with using your application from the computer. To improve it, add additional commands and extend the protocol.

11. Add a help message in case of invalid messages or received letter **h** | **H**.
12. Add an additional command to get the current state of the LED.
13. Add to the message parser a new command with setting the state of an color. The state will be defined after a colon (:) with the values 0 and 1. As an end token, to recognise a full message, the character LF (newline (`'\n'`)) is used.

> **Note**
>
> With an end token, a complete message is defined as with UART an message can only read byte per byte. Generally, end tokens are defined by the protocol with an character like CR (`'\r'`), LF (`'\n'`) or both together.

- **r\n** | **R\n** - Toggles the red colour channel
- **g\n** | **G\n** - Toggles the green colour channel
- **b\n** | **B\n** - Toggles the blue colour channel
- **<COLOR>:<STATE>\n** - Sets the colour to the desired state, e.g:
    - r:1\n - Turn ON the RED LED
    - B:0\n - Turn OFF the BLUE LED