Lab 6: Universal Asynchronous Receiver-Transmitter (UART)

Learning objectives

After completing this lab you will be able to:

- Understand the UART communication
- Decode UART frames
- Use functions from UART library
- Use logic analyzer

The purpose of the laboratory exercise is to understand serial asynchronous communication, data frame structure and communication options using an internal USART unit.

Table of contents

Components list

- Arduino Uno board, USB cable
- Logic analyzer

Pre-Lab preparation

1. Use schematic of Arduino Uno board and find out on which Arduino Uno pins the UART transmitter (Tx) and receiver (Rx) are located.

2. Remind yourself, what an ASCII table is. What codes are defined for control characters `Esc`, `Space`, `Tab`, and `Enter`?

Part 1: Basics of UART communication

The UART (Universal Asynchronous Receiver-Transmitter) is not a communication protocol like SPI and I2C, but a physical circuit in a microcontroller, or a stand-alone integrated circuit, that translates communicated data between serial and parallel forms. It is one of the simplest and easiest method for implement and understanding.

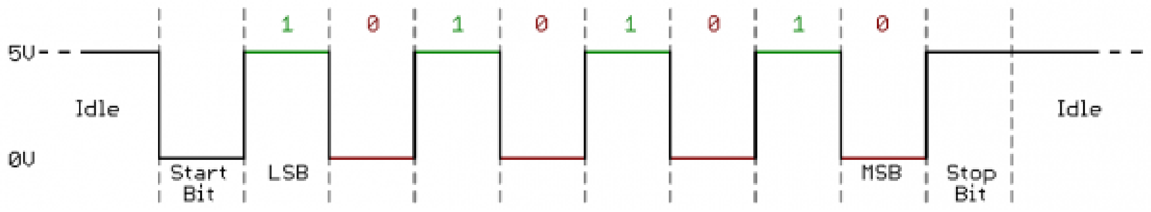
In UART communication, two UARTs communicate directly with each other. The transmitting UART converts parallel data from a CPU into serial form, transmits it in serial to the receiving UART, which then converts the serial data back into parallel data for the receiving device. Only two wires are needed to transmit data between two UARTs. Data flows from the Tx pin of the transmitting UART to the Rx pin of the receiving UART [3], [4].

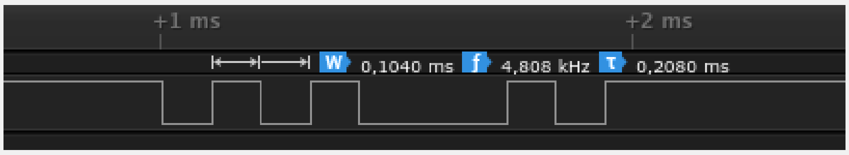
UARTs transmit data asynchronously, which means there is no external clock signal to synchronize the output of bits from the transmitting UART. Instead, timing is agreed upon in advance between both units, and special **Start** (log. 0) and 1 or 2 **Stop** (log. 1) bits are added to each data package. These bits define the beginning and end of the data packet so the receiving UART knows when to start reading the bits. In addition to the start and stop bits, the packet/frame also contains data bits and optional parity.

The amount of **data** in each packet can be set from 5 to 9 bits. If it is not otherwise stated, data is transferred least-significant bit (LSB) first.

Parity is a form of very simple, low-level error checking and can be Even or Odd. To produce the parity bit, add all 5-9 data bits and extend them to an even or odd number. For example, assuming parity is set to even and was added to a data byte `0110_1010`, which has an even number of 1's (4), the parity bit would be set to 0. Conversely, if the parity mode was set to odd, the parity bit would be 1.

One of the most common UART formats is called **9600 8N1**, which means 8 data bits, no parity, 1 stop bit and a symbol rate of 9600 Bd.



> Example of UART communication
>
> **Question:** Let the following image shows one frame of UART communication transmitting from the ATmega328P in 8N1 mode. What ASCII code/character does it represent? According to bit period, estimate the symbol rate.
>
> 
>
>
> **Answer:** 8N1 means that 8 data bits are transmitted, no parity is used, and the number of stop bits is one. Because the frame always starts with a low level start bit and the order of the data bits is from LSB to MSB, the data transmitted bu UART is therefore `0100_0101` (0x45) and according to the ASCII (American Standard Code for Information Interchange) table, it represents the letter `E`.
>
> The figure further shows that the bit period, i.e. the duration of one bit, is 104 us. The symbol rate of the communication is thus 1/104e-6 = 9615, i.e. approximately 9600 Bd.

Part 2: Synchronize repositories and create a new project

1. In your working directory, use **Source Control (Ctrl+Shift+G)** in Visual Studio Code or Git Bash (on Windows) or Terminal (on Linux) to update the local repository.

   > **Help:** Useful bash and git commands are `cd` - Change working directory. `mkdir` - Create directory. `ls` - List information about files in the current directory. `pwd` - Print the name of the current working directory. `git status` - Get state of working directory and staging area. `git pull` - Update local repository and working folder.

2. In Visual Studio Code create a new PlatformIO project `lab6-uart` for `Arduino Uno` board and change project location to your local repository folder `Documents/digital-electronics-2`.

3. IMPORTANT: Rename `LAB6-UART > src > main.cpp` file to `main.c`, ie change the extension to `.c`.

Part 3: Communication between Arduino board and computer

In the lab, we are using UART library developed by Peter Fleury.

1. Use online manual of UART library and add input parameters and description of the following functions.

Function name	**Function parameter(s)**	**Description**	**Example**
`uart_init`	`UART_BAUD_SELECT(9600, F_CPU)`	Initialize UART to 8N1 and set baudrate to 9600 Bd	`uart_init(UART_BAUD_SELECT(9600, F_CPU));`
`uart_getc`			
`uart_putc`			
`uart_puts`			

2. Copy/paste template code to `LAB6-UART > src > main.c` source file.

3. Use your favorite file manager and copy `timer.h` file from the previous labs to `LAB6-UART > include` folder.

4. In PlatformIO project, create a new folder `LAB6-UART > lib > uart`. Within this folder, create two new files `uart.c` and `uart.h`.

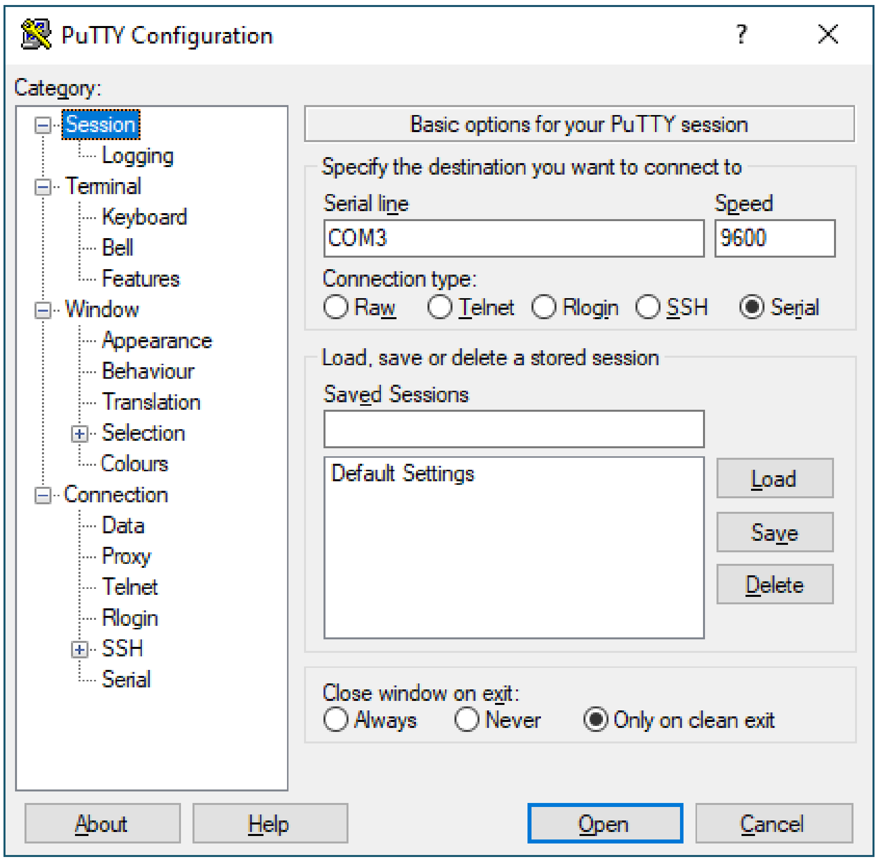
   1. Copy/paste library source file to `uart.c`
   2. Copy/paste header file to `uart.h`

   The final project structure should look like this:

```
LAB6-UART            // PlatfomIO project
├── include          // Included file(s)
│   └── timer.h
├── lib              // Libraries
│   └── uart         // Peter Fleury's UART library
│       ├── uart.c
│       └── uart.h
├── src              // Source file(s)
│   └── main.c
├── test             // No need this
└── platformio.ini   // Project Configuration File
```

5. Go through the `main.c` file and make sure you understand each line. Build and upload the code to Arduino Uno board. What is the meaning of ASCII control characters `\r`, `\n`, and `\t`?

Use **PlatformIO: Serial Monitor** or **PuTTY application** to receive values from Arduino board. In PuTTY, set connection type to `Serial` and check that configuration is the same as in the ATmega328P application, ie. 9600 8N1 mode. Note that, **serial line** (here COM3 on Windows) could be different. In Linux, use `dmesg` command to verify your port (such as `/dev/ttyUSB0`).



Basic options for your PuTTY session

Specify the destination you want to connect to

Serial line: COM3

Speed: 9600

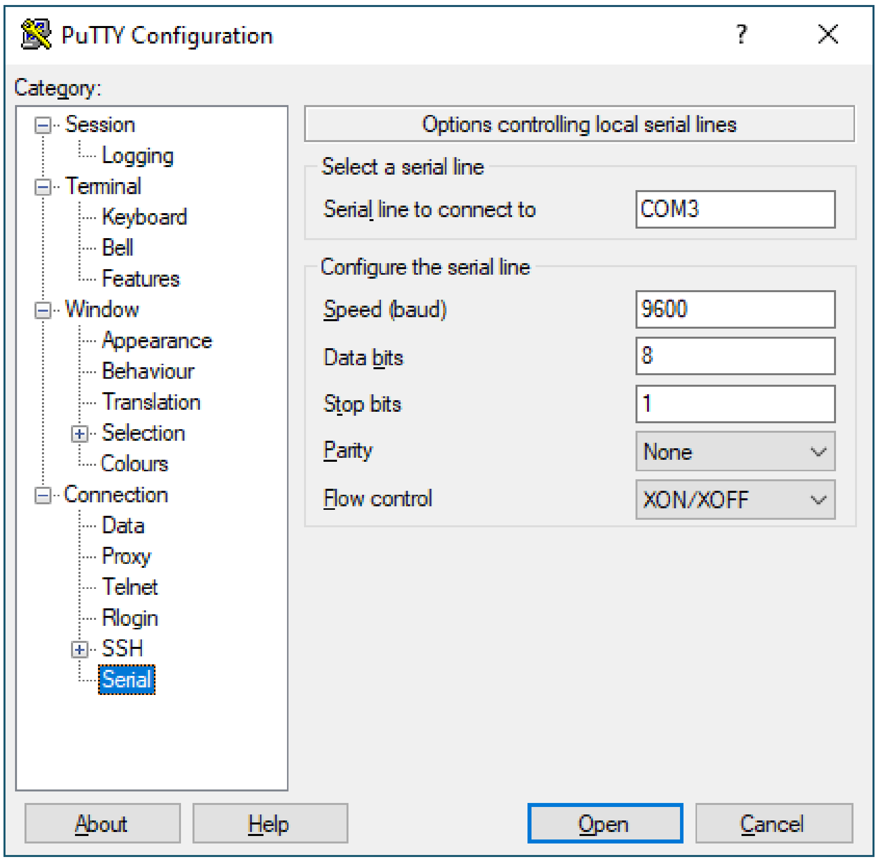
Connection type: Raw Telnet Rlogin SSH [✓ Serial]

Load, save or delete a stored session

Saved Sessions

Default Settings

Close window on exit: Always Never [✓ Only on clean exit]



Options controlling local serial lines

Select a serial line

Serial line to connect to: COM3

Configure the serial line

Speed (baud): 9600

Data bits: 8

Stop bits: 1

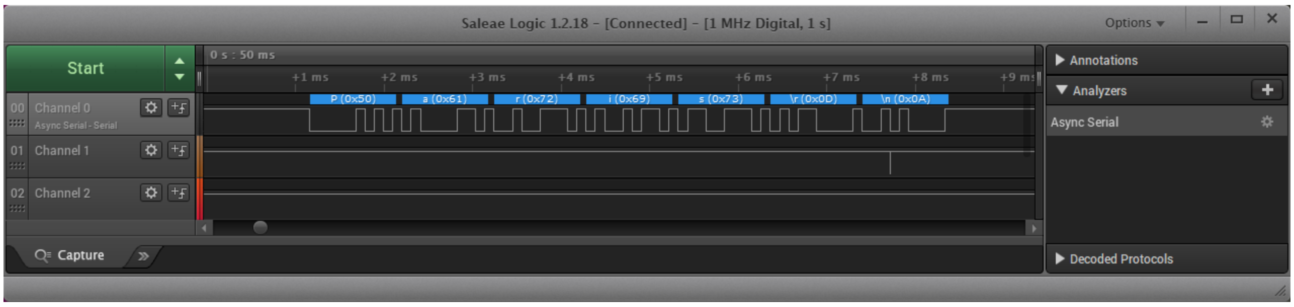
Parity: None

Flow control: XON/XOFF

> **Warning:** Before Arduino board re-programming process, PuTTY app must be closed!

> In SimulIDE, right click to ATmega328 package and select **Open Serial Monitor**. In this window you can receive data from the microcontroller, but also send them back.

6. Configure Timer1 to overflow four times per second and transmit UART string `Paris`. Use Logic Analyzer to visualize and decode transmitting strings.



> **Note:** You have to have a Saleae logic analyzer or similar, and to download and install Saleae Logic 1 software on your computer.
>
> Tutorial about using a logic analyzer is available in this video.

7. Use `uart_getc` function and display the ASCII code of received character in decimal, hexadecimal, and binary. You can use Timer1 overflow handler to perform such receiver. Fill the table with selected keys.

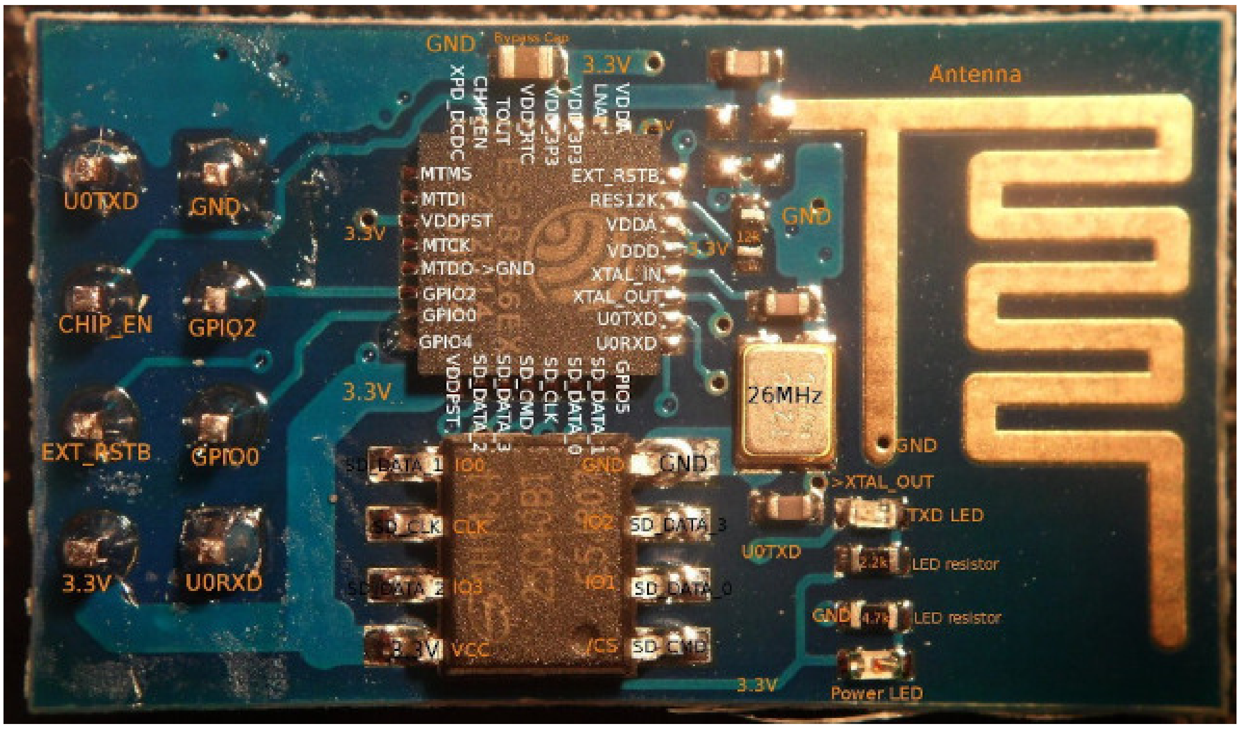
```c
ISR(TIMER1_OVF_vect)
{
    uint8_t value;
    char string[8];  // String for converted numbers by itoa()

    value = uart_getc();
    if (value != '\0') {  // Data available from UART
        // Display ASCII code of received character
        // WRITE YOUR CODE HERE
    }
}
```

Char	Decimal	Hexadecimal	Binary
`a`	97	0x61	`0b0110_0001`
`b`			
`c`			
`0`	48	0x30	`0b0011_0000`
`1`			
`2`			
`Esc`			
`Space`			
`Tab`			

Char	Decimal	Hexadecimal	Binary
`Backspace`			
`Enter`			

8. (Optional) Verify basic AT commands of Wi-Fi module ESP8266 ESP-01. Connect Wi-Fi module to Arduino Uno board according to the following instructions.

ESP-01 pin	Arduino Uno pin	ESP-01 pin	Arduino Uno pin
U0TXD	Tx (pin 1)	GND	GND
CHIP_EN	3.3V	GPIO2	Not connected
EXT_RSTB	Not connected	GPIO0	Not connected
3.3V	3.3V	U0RXD	Rx (pin 0)



In your code, disable interruptions by commenting `// sei();` function. The reason is the micro controller will not affect UART lines and whole communication will be done between Serial Monitor and Wi-Fi module. To use PlatfomIO Serial Monitor, add the following command to `platformio.ini` project configuration file: `monitor_speed = 115200`. Compile and upload the application. Test the following AT commands and see the module's responses. If needed, use Logic analyzer to read the response of Wi-Fi module.

- `AT` - Check the communication with module
- `AT+CWMODE=1` - Set the module mode
- `AT+GMR` - Get the module version
- `AT+CWLAPOPT=1,6` - Limit the list to `rssi` and `ssid` parameters only
- `AT+CWLAP` - List `ssid` and `rssi` parameters of available Wi-Fi APs (takes few seconds)

The complete list and description of all AT commands are available here. To avoid a conflict with Wi-Fi module, remove the Tx and Rx wires when uploading the firmware and put them back after the upload is complete.

9. After completing your work, ensure that you synchronize the contents of your working folder with both the local and remote repository versions. This practice guarantees that none of your changes are lost. You can achieve this by using **Source Control (Ctrl+Shift+G)** in Visual Studio Code or by utilizing Git commands.

> **Help:** Useful git commands are `git status` - Get state of working directory and staging area. `git add` - Add new and modified files to the staging area. `git commit` - Record changes to the local repository. `git push` - Push changes to remote repository. `git pull` - Update local repository and working folder. Note that, a brief description of useful git commands can be found here and detailed description of all commands is here.

(Optional) Experiments on your own

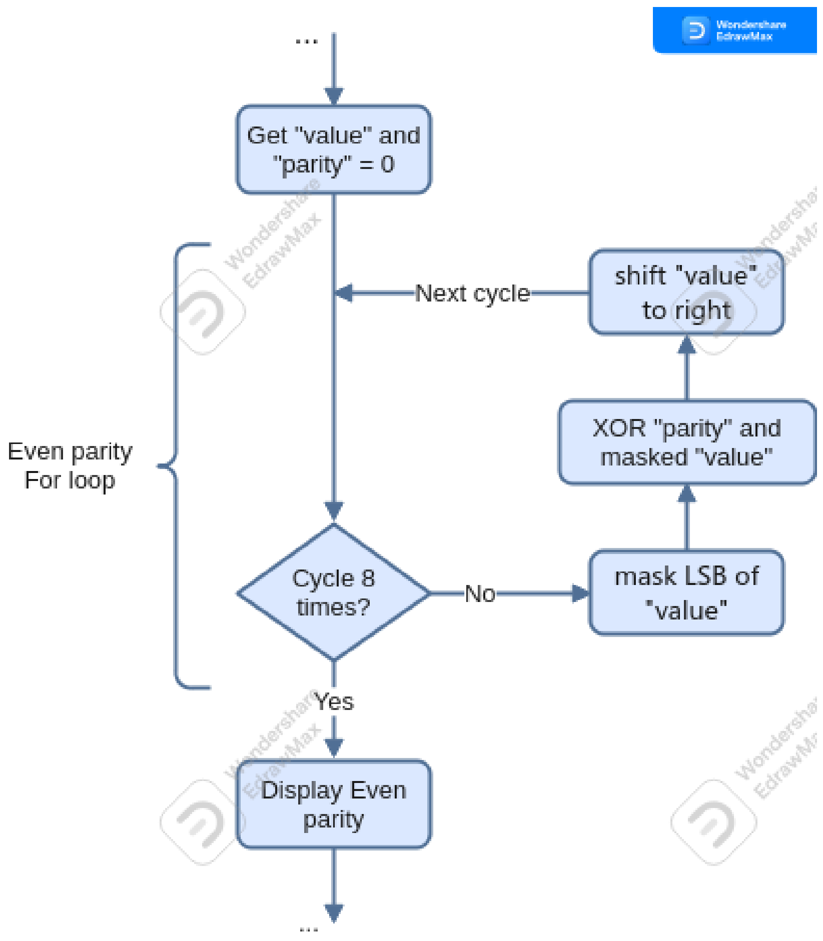
1. Use ANSI Escape Sequences and modify color and format of transmitted strings according to the following code. Try other formatting styles.

```
/*
 * Color/formatting sequence is prefixed with `Escape` (`\x1b` in
hexadecimal,
 * \033, etc), followed by opening square bracket `[`, commands delimited
by
 * semi colon `;` and ended by `m` character.
 *
 * Examples:
 *   \x1b[1;31m  - Set style to bold, red foreground
 *   \x1b[4,32m  - Set underline style, green foreground
 *   \x1b[0m     - Reset all attributes
 */
uart_puts("\x1b[4;32m");  // 4: underline style; 32: green foreground
uart_puts("This is all Green and Underlined\r\n");
uart_puts("\x1b[0m");     // 0: reset all attributes
uart_puts("This is Normal text again\r\n");
```

To enable ANSI color codes in PlatformOI serial monitor, add the following line to `platformio.ini` project configuration file.

```
# Enable ANSI color codes in serial monitor
monitor_raw = yes
```

2. Program a piece of code to calculate even parity bit from received value.



3. Draw a timing diagram of the output from UART/USART when transmitting three character data `De2` in 4800 7O2 mode (7 data bits, odd parity, 2 stop bits, 4800 Bd). The image can be drawn on a computer (by WaveDrom for example) or by hand. Name all parts of timing diagram.

4. Program a software UART transmitter (emulated UART) that will be able to generate UART data on any output pin of the ATmega328P microcontroller. Let the bit rate be approximately 9600 Bd and do not use the delay library. Also consider the possibility of calculating the parity bit. Verify the UART communication with logic analyzer or oscilloscope.

5. Program an interactive console that communicates between ATmega328P and the computer (PuTTY application) via UART. Let the main screen of the console is as follows:

   ```
   --- Interactive UART console ---
   1: read current Timer/counter1 value
   2: reset Timer/counter1
   >
   ```

   After pressing the '1' key on computer keyboard, ATmega328P receives ASCII code of the key and sends the current Timer1 value back to PuTTY. After pressing the '2' key, ATmega328P resets Timer1 value, etc. Use ANSI escape sequences to highlight information in PuTTY console.

   ```c
   uint8_t value;
   ...
   value = uart_getc();
   if (value != '\0') {     // Data available from UART
   ```

```
        if (value == `1`) {  // Key `1` received
            ...
        }
    }
    ...
```

> **Warning:** Keep UART strings as short as possible. But if you need to transmit a larger amount of data, it is necessary to increase the size of the transmit/receive buffer in the `uart.h` file, eg to 128.
>
> ```
> /** @brief  Size of the circular receive buffer, must be power of 2
>  *
>  *  You may need to adapt this constant to your target and your application by adding
>  *  CDEFS += -DUART_RX_BUFFER_SIZE=nn to your Makefile.
>  */
> #ifndef UART_RX_BUFFER_SIZE
> # define UART_RX_BUFFER_SIZE 128
> #endif
>
> /** @brief  Size of the circular transmit buffer, must be power of 2
>  *
>  *  You may need to adapt this constant to your target and your application by adding
>  *  CDEFS += -DUART_TX_BUFFER_SIZE=nn to your Makefile.
>  */
> #ifndef UART_TX_BUFFER_SIZE
> # define UART_TX_BUFFER_SIZE 128
> #endif
> ```

6. Finish all (or several) experiments, upload them to your GitHub repository, and submit the project link via BUT e-learning. The deadline for submitting the assignment is the day prior to the next lab session, which is one week from now.

References

1. Tomas Fryza. Schematic of Arduino Uno board

2. ASCII Table

3. Circuit Basics. Basics of UART Communication

4. Eric Peña, Mary Grace Legaspi. UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter

5. Peter Fleury. UART library

6. Kolban Technical Tutorials. ESP32 Technical Tutorials: Using a logic analyzer

7. Tomas Fryza. Useful Git commands

8. Christian Petersen. ANSI Escape Sequences