UNIVERSITATEA TEHNICĂ A MOLDOVEI

FACULTATEA: CALCULATOARE, INFORMATICĂ ŞI MICROELECTRONICĂ

DEPARTAMENTUL: INGINERIA SOFTWARE ȘI AUTOMATICA

Laboratory work NR. 1.1

Interaction with the user: Serial, STDIO

Executed: Konjevic Alexandra, gr. FAF-213

Verified: Assistant Professor Moraru Dumitru

Chișinău – 2024

Task: Creating applications for interaction with user through serial interface, using STDIO library, to be able to use functions `printf()` and `scanf()`.

Objectives:

<u>Task 1.</u> Configure application for working with the library STDIO through serial interface for exchanging text in the terminal.

<u>Task 2.</u> Create an MCU-based application which takes instructions from terminal (using serial interface) for setting the state of a led:

- `led on` for turning the led on
- `led off` for turning it off
- system needs to respond with text messages for confirming the commands
- for exchanging the text through terminal, the library STDIO must be used

Implementation:

Given the description of the tasks, we can understand that the main objective is understanding how to interact with a user through serial monitor. The Serial Monitor is an essential tool when creating projects with Arduino. It can be used as a debugging tool, testing out concepts or to communicate directly with the Arduino board. We can to create a sketch that includes some configurations needed, so that our board can communicate with our computer. Mainly, we need to set a baud rate, which is done by writing `Serial.begin(9600)`. Here, the `9600` represents the baud rate, which is the maximum bits per seconds that can be transferred. The sketch that we need to use can be found in the snippet below (This will print "Hello world!" to the Serial Monitor):

```
void setup() {
  Serial.begin(9600);
}
void loop() {
  Serial.println("Hello world!");
}
```

Now, let's proceed to the implementation. First of all, I assembled the needed part of the system in Proteus, to be able to simulate this system. The scheme that I've obtained is represented in the figure 1:

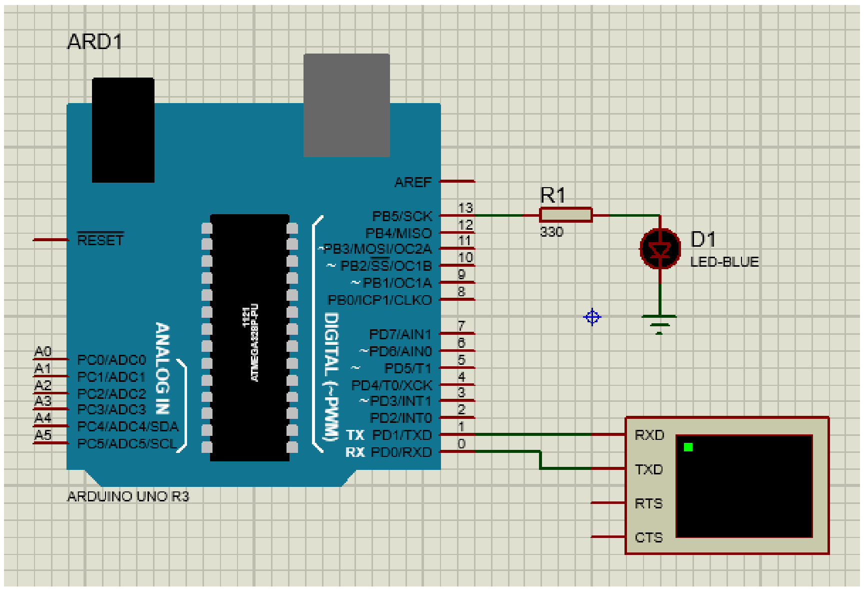


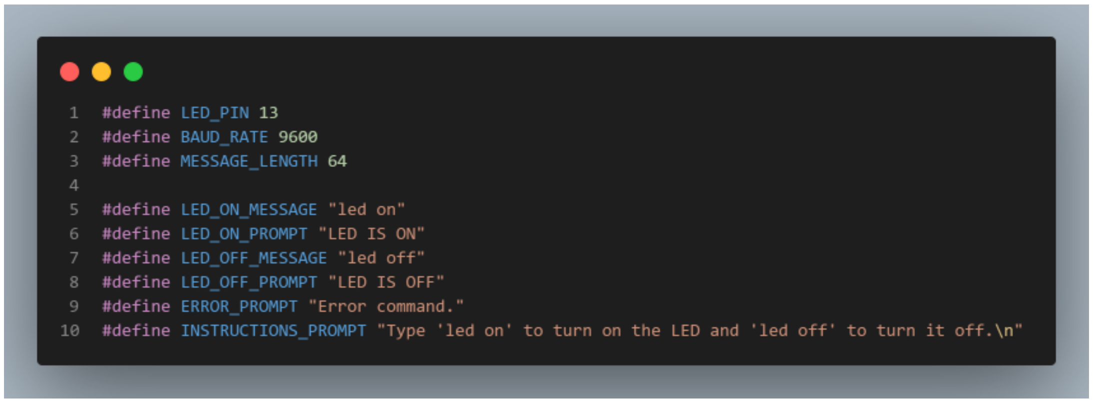
Figure 1. Simulated electrical schema

There you can see the devices that I used for this simulation: an Arduino UNO R3 board, a led of blue color, a resistor and a virtual terminal.

Virtual terminal is a useful tool available in Proteus, that helps to simulate the serial communication of a user with the Arduino board. The virtual terminal in Proteus is bi-directional, which means that it can send and receive data simultaneously. So, virtual terminal is used to view data coming from Serial Port and also used to send the data to Serial Port. Virtual terminal has four pins: two are for data transmitting and receiving and other two terminals represent the handshaking between the components to be communicated. As shown in the scheme above, the receiving pin of the virtual terminal is connected to the transmitting pin of the Arduino, and, vice versa, the transmitting pin of virtual terminal is connected to the receiving pin of the Arduino. When trying to simulate the mechanism in Proteus, the terminal window will be shown, and inside it, we can type the commands and also see the output of the Arduino.

Now, moving to the implementation of the program, I will emphasize the program part by part:

1. Libraries used: I used the `Arduino.h` and `stdio.h` libraries. The `stdio.h` is used for printing the output to the serial monitor, using the function `printf`; I also use the functions `memset` and `strcmp` from this library.
2. Definition of the constants:

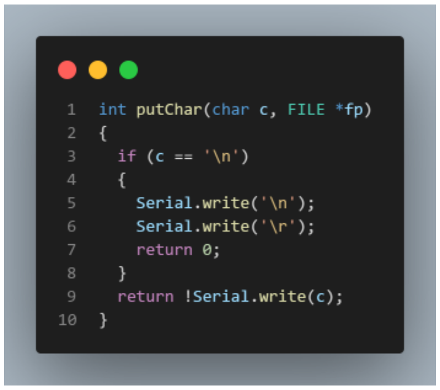


```
#define LED_PIN 13
#define BAUD_RATE 9600
#define MESSAGE_LENGTH 64

#define LED_ON_MESSAGE "led on"
#define LED_ON_PROMPT "LED IS ON"
#define LED_OFF_MESSAGE "led off"
#define LED_OFF_PROMPT "LED IS OFF"
#define ERROR_PROMPT "Error command."
#define INSTRUCTIONS_PROMPT "Type 'led on' to turn on the LED and 'led off' to turn it off.\n"
```

Figure 2. Definition of constants in the program

The `LED_ON_MESSAGE` and `LED_OFF_MESSAGE` are used to define the command that needs to be introduced to change the state of the led, whereas the `LED_ON_PROMPT` and `LED_OFF_ PROMPT` are used to show the output on the serial monitor, when the state is changed.

3. `putChar` function:

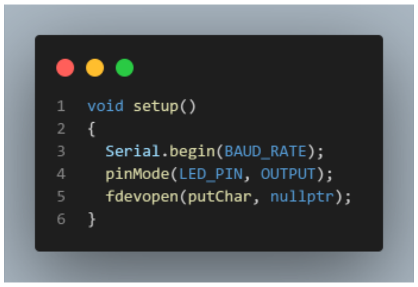


```
int putChar(char c, FILE *fp)
{
  if (c == '\n')
  {
    Serial.write('\n');
    Serial.write('\r');
    return 0;
  }
  return !Serial.write(c);
}
```

Figure 3. `putChar` function

This is a function that I used to be able to use the `stdio.h` library's function ` printf`. This function specifies a target for STDOUT. If a device such as a terminal needs special handling, it is in the domain of the terminal device driver to provide this functionality. Thus, a simple function suitable as `put()` for `fdevopen()` that talks to a UART interface might look like the one that I implemented: `putChar()`. The function passed as put shall take two arguments, the first a character to write to the device, and the second a pointer to FILE, and shall return 0 if the output was successful.

4. Setup:

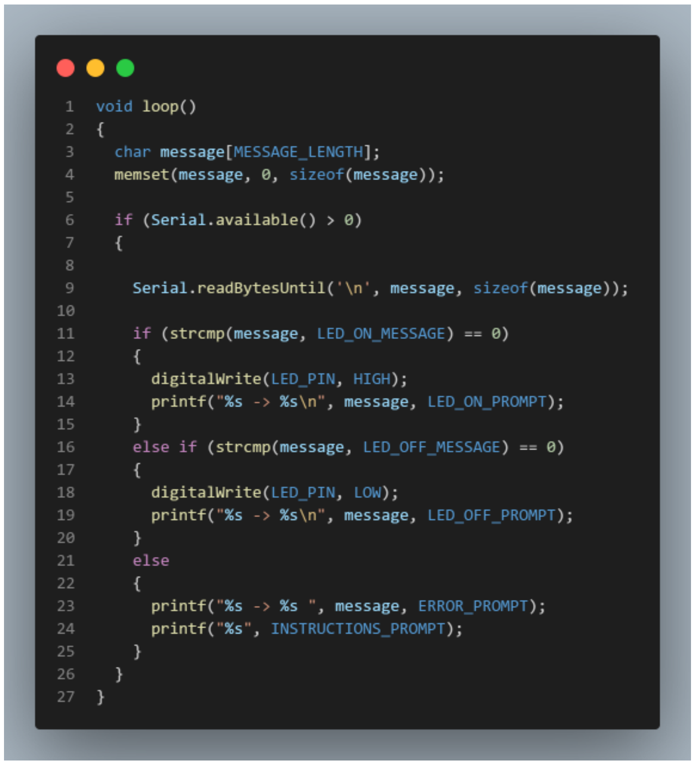


```
void setup()
{
  Serial.begin(BAUD_RATE);
  pinMode(LED_PIN, OUTPUT);
  fdevopen(putChar, nullptr);
}
```

Figure 4. Setup function of the program

Here I have the command `Serial.begin(9600)`, which starts serial communication, so that the Arduino can send out commands. Next, the `pinMode()` function configures the `LED_PIN` (13) to behave as an output. Next, I use the function `fdevopen()`. It opens a stream for a device where the actual device implementation needs to be provided by the application. If successful, a pointer to the structure for the opened stream is returned. If the put function pointer (I use `putChar` pointer here) is provided, the stream is opened with write intent.

5. Loop:

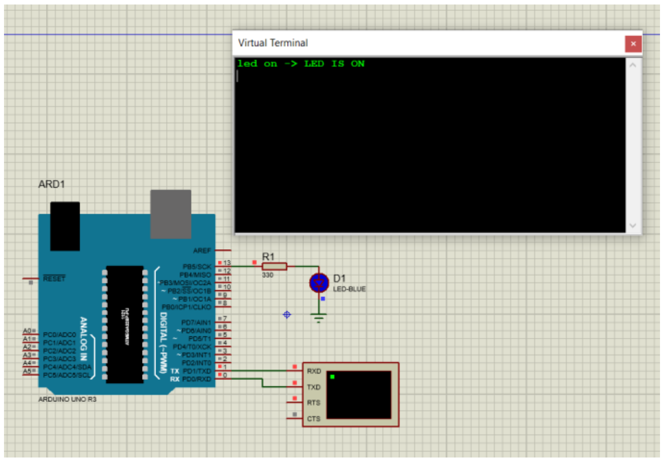


```
void loop()
{
  char message[MESSAGE_LENGTH];
  memset(message, 0, sizeof(message));

  if (Serial.available() > 0)
  {

    Serial.readBytesUntil('\n', message, sizeof(message));

    if (strcmp(message, LED_ON_MESSAGE) == 0)
    {
      digitalWrite(LED_PIN, HIGH);
      printf("%s -> %s\n", message, LED_ON_PROMPT);
    }
    else if (strcmp(message, LED_OFF_MESSAGE) == 0)
    {
      digitalWrite(LED_PIN, LOW);
      printf("%s -> %s\n", message, LED_OFF_PROMPT);
    }
    else
    {
      printf("%s -> %s ", message, ERROR_PROMPT);
      printf("%s", INSTRUCTIONS_PROMPT);
    }
  }
}
```

Figure 5. Loop function of the program

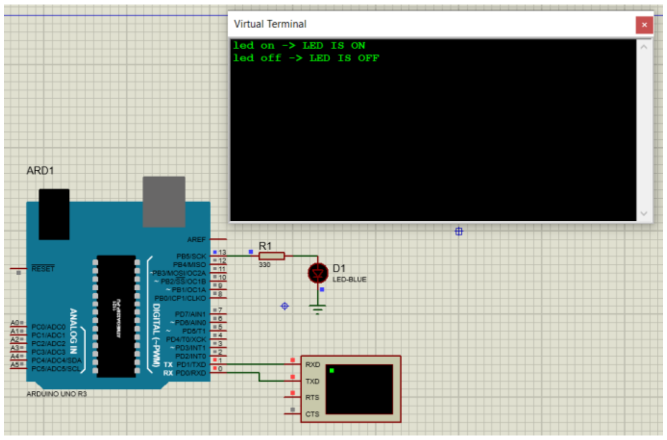
In the main loop of the program, I firstly declared the `message` variable, which represents the message introduced by the user (assuming it won't exceed the length of 64 characters), and I initialized this variable with zero at all indexes. After that, I checked if there are any characters available to read from the serial port – `Serial.available()` returns the number of available characters, thus, if its values if greater than zero, there are characters. Next, using the function ` Serial.readBytesUntil()`, the program reads the input introduced by the user until they press the enter key. After that, this message read from the input, is compared with the commands `led on` and `led off`. If it is either of that command, the led does the corresponding action, but if it is not, the output "Error command. Type 'led on' to turn on the LED and 'led off' to turn it off." is shown.

Simulation:



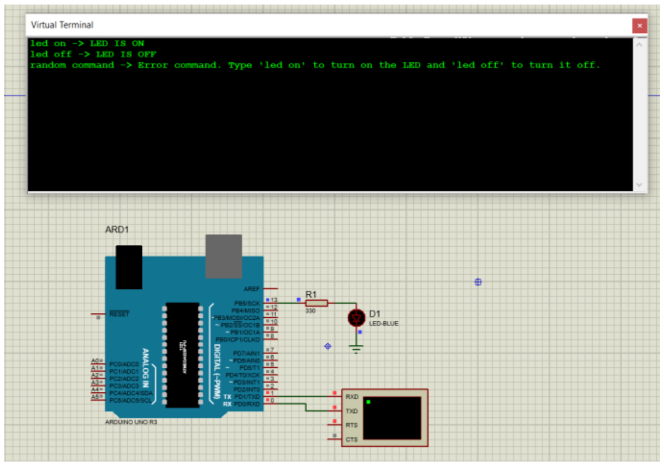
```
led on -> LED IS ON
```

Figure 6. "led on" command



```
led on -> LED IS ON
led off -> LED IS OFF
```

Figure 7. "led off" command



```
led on -> LED IS ON
led off -> LED IS OFF
random command -> Error command. Type 'led on' to turn on the LED and 'led off' to turn it off.
```

Figure 8. Random command

Conclusion:

In conclusion, the development and implementation of the Arduino circuit with a virtual terminal have proven to be a successful and valuable experience. The integration of a virtual terminal for communication via the serial monitor enhances the versatility and usability of the circuit, opening up a wide range of possibilities for real-time data exchange and control.

Through the course of this project, I have gained a deeper understanding of Arduino programming, serial communication protocols, and the practical application of virtual terminals. The circuit's ability to establish seamless communication between the Arduino board and external devices or software provides a robust platform for various projects and applications.

Moreover, the virtual terminal's role in facilitating debugging and monitoring has significantly improved the overall development process. The real-time feedback and data visualization offered by the terminal not only streamline troubleshooting but also enhance the efficiency of the circuit during both development and operational phases.

This project has underscored the importance of effective communication interfaces in embedded systems, and the virtual terminal serves as a testament to the adaptability and user-friendly nature of Arduino-based projects. As technology continues to evolve, the skills and knowledge acquired through this project will undoubtedly contribute to future endeavors in the field of electronics and embedded systems.

In conclusion, the successful realization of the Arduino circuit with a virtual terminal not only meets the initial project objectives but also opens the door to endless possibilities for innovation and exploration in the exciting realm of Arduino development.

BIBLIOGRAPHY

1. Arduino: Arduino UNO R3. Arduino official site, ©2024 [quote 2024]. Access link: https://docs.arduino.cc/hardware/uno-rev3/
2. Serial Monitor: Using the Serial Monitor tool. Arduino official site, ©2024 [quote 2024]. Access link: https://docs.arduino.cc/software/ide-v2/tutorials/ide-v2-serial-monitor/
3. Virtual Terminal: Virtual Terminal in proteus. ProjectIOT article, @2024 [quote 03.04.2019]. Access link: https://projectiot123.com/2019/04/03/virtual-terminal-in-proteus/
4. \`printf()\`: printf in Arduino works. Reza's Rants article, @2016 [quote 21.12.2010]. Access link: http://reza.net/wordpress/?p=269
5. \`fdevopen()\`: <stdio.h>: Standard IO facilities. Nongnu article, @2024 [quote 29.01.2022]. Access link: https://www.nongnu.org/avr-libc/user-manual/group__avr__stdio.html