Федеральное государственное автономное образовательное
учреждение высшего образования
«Национальный исследовательский университет ИТМО»

Дисциплина: Проектирование вычислительных систем

Лабораторная работа 2

Вариант 2

Выполнили:
Колбасин Владислав Ильич,
Митичев Иван Дмитриевич,
группа P3416

Преподаватель:
Пинкевич Василий Юрьевич

2025 г.
Санкт-Петербург

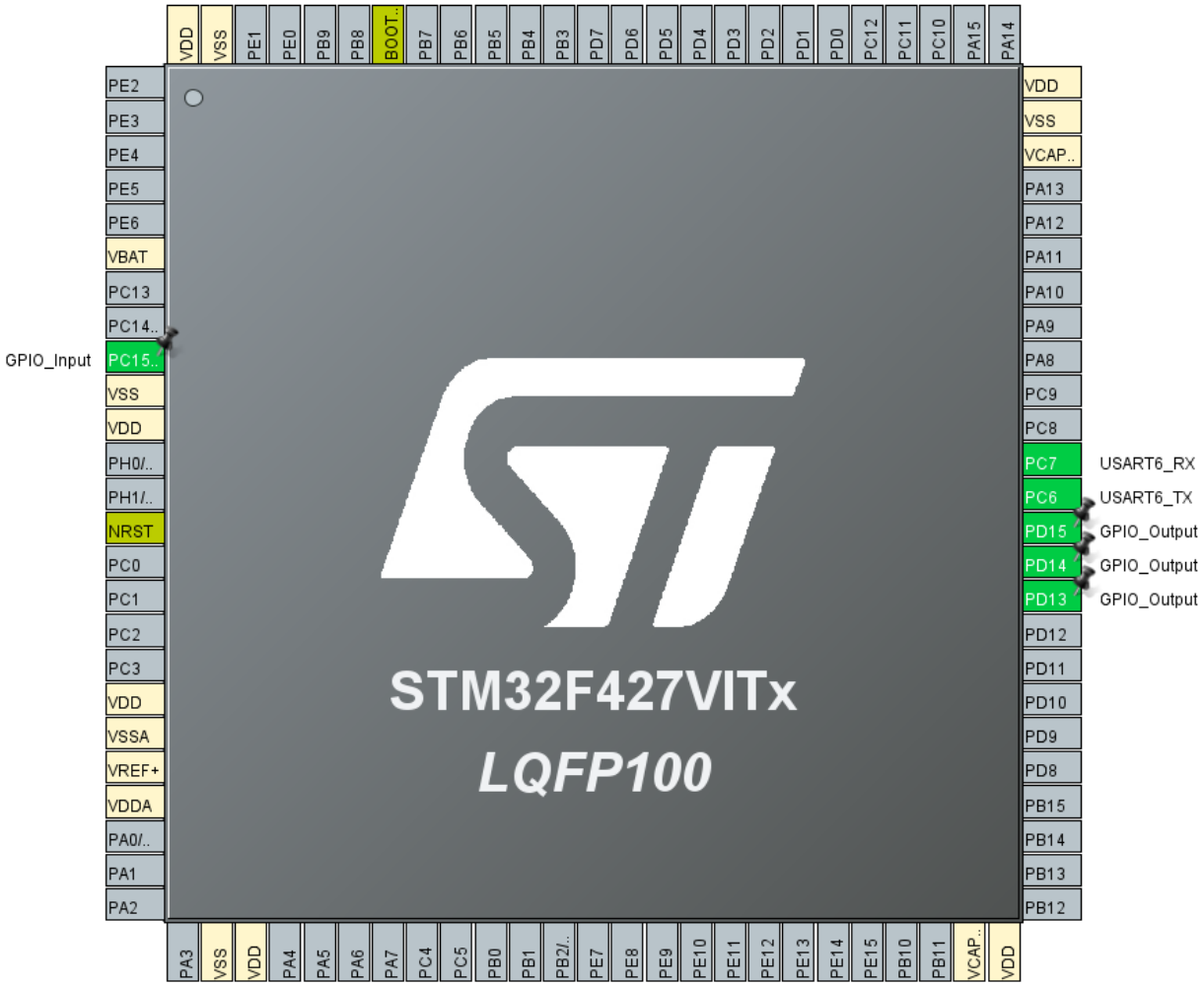
Задание

Разработать и реализовать два варианта драйверов UART для стенда SDK-1.1M: с использованием и без использования прерываний. Драйверы, использующие прерывания, должны обеспечивать работу в «неблокирующем» режиме (возврат из функции происходит сразу же, без ожидания окончания приема/отправки), а также буферизацию данных для исключения случайной потери данных. В драйвере, не использующем прерывания, функция 77 приема данных также должна быть «неблокирующей», то есть она не должна зависать до приема данных (которые могут никогда не поступить). При использовании режима «без прерываний» прерывания от соответствующего блока UART должны быть запрещены.

Написать с использованием разработанных драйверов программу, которая выполняет определенную вариантом задачу. Для всех вариантов должно быть реализовано два режима работы программы: с использованием и без использования прерываний. Каждый принимаемый стендом символ должен отсылаться обратно, чтобы он был выведен в консоли (так называемое «эхо»). Каждое новое сообщение от стенда должно выводиться с новой строки. Если вариант предусматривает работу с командами, то на каждую команду должен выводиться ответ, определенный в задании или «ОК», если ответ не требуется. Если введена команда, которая не поддерживается, должно быть выведено сообщение об этом.

Скорость работы интерфейса UART должна соответствовать указанной в варианте задания.

Используемые контакты

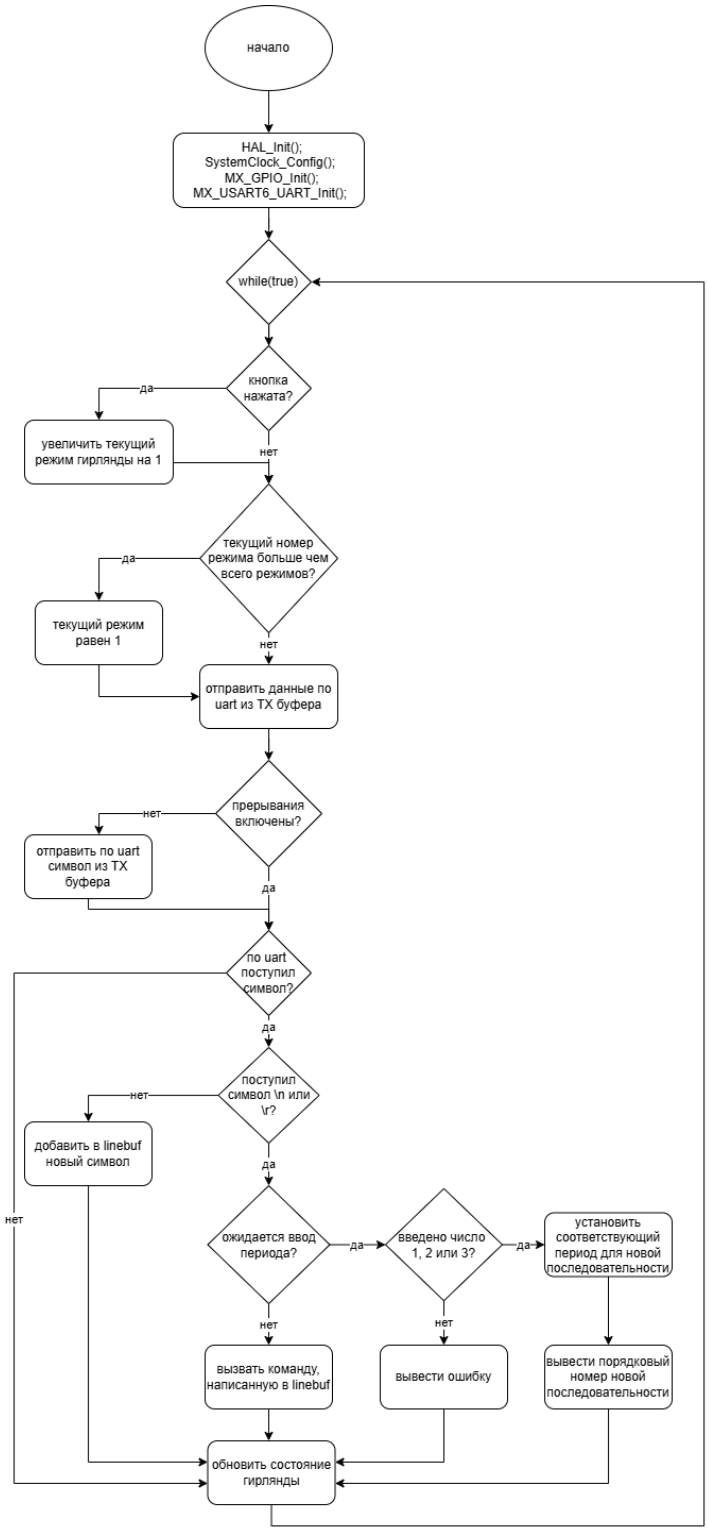


Описание контактов

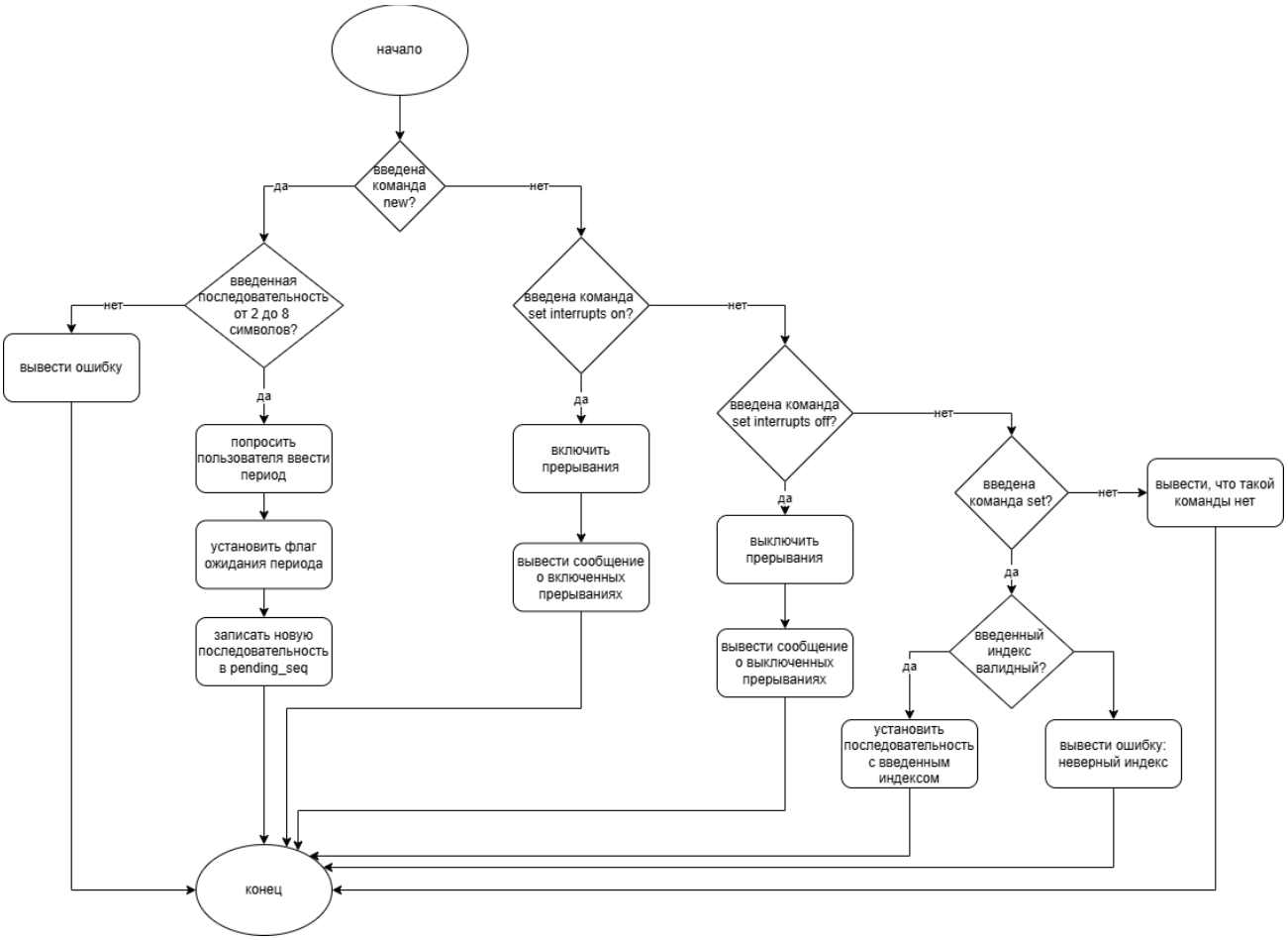
- **PC15** - GPIO_Input (кнопка)
- **PD13** - GPIO_Output (зеленый светодиод)
- **PD14** - GPIO_Output (желтый светодиод)
- **PD15** - GPIO_Output (красный светодиод)
- **PC6** – USART6_TX
- **PC7** – USART6_RX

Блок-схемы

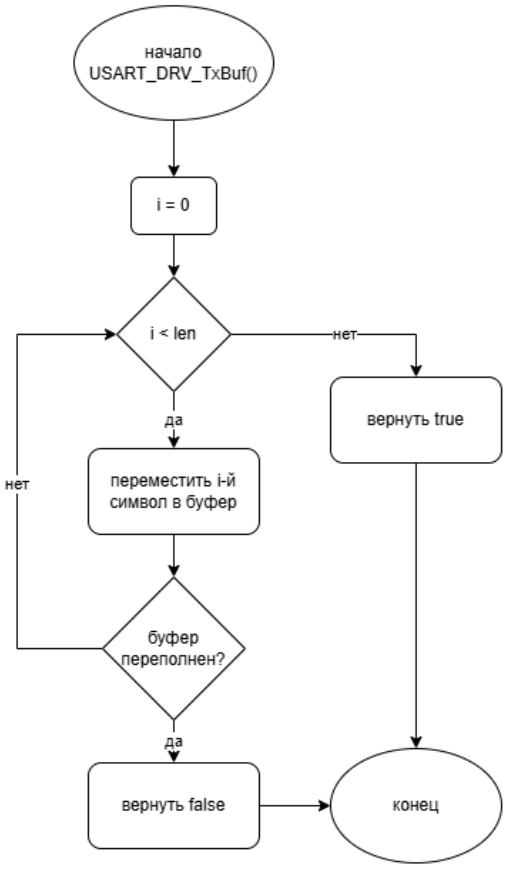
Основная программа:

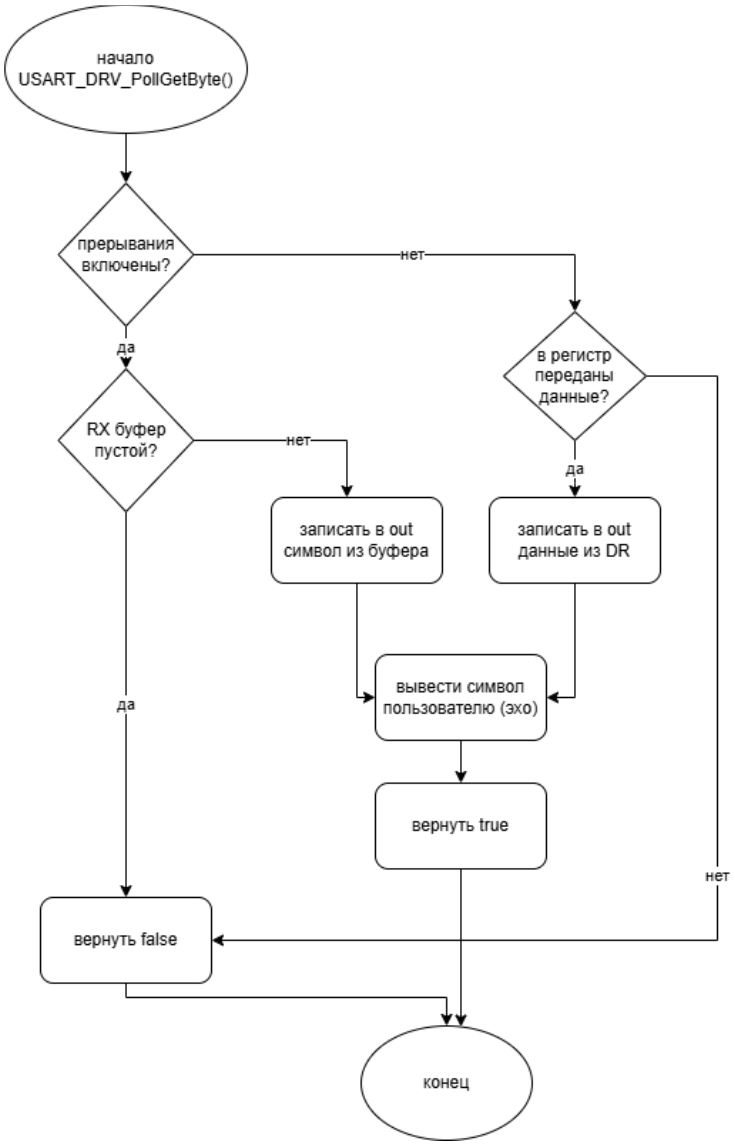


Подпрограмма выполнения команд:



UART драйвер:





Код программы

main.c:

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2025 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "usart.h"
#include "gpio.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "led_driver.h"
#include "button_driver.h"
#include "usart_drv.h"
#include "stm32f4xx_hal.h"
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
typedef struct {
    uint8_t seq[8]; // последовательность битов: 0->none, 1->g,2->r,4->y (только 1 бит установлен)
    uint8_t len;
    uint32_t period_ms;
} seq_t;
/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define MAX_PRESET 8
#define CMD_BUF_SZ 64
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */
```

```c
/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/

/* USER CODE BEGIN PV */
 static seq_t presets[MAX_PRESET+1]; // индекс 1..8
 static int active_seq = 1;
 static uint8_t seq_index[9]; // позиция в каждой seq
 static int next_user_slot = 5; // куда сохранять new (5..8 cyclic)
 static int total_presets = 4;

 static bool awaiting_period = false;
 static char pending_seq[8];
 static int pending_len = 0;

 static char linebuf[CMD_BUF_SZ];
 static int linepos = 0;
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
static uint8_t char_to_mask(char c)
{
    if(c == 'g' || c == 'G') return 0x1;
    if(c == 'y' || c == 'Y') return 0x2;
    if(c == 'r' || c == 'R') return 0x4;
    return 0x0;
}

/* инициализация предустановленных 1..4 */
static void init_presets_defaults(void)
{
    presets[1].len = 3; presets[1].seq[0]=0x1; presets[1].seq[1]=0x2;
presets[1].seq[2]=0x4; presets[1].period_ms=300;
    presets[2].len = 2; presets[2].seq[0]=0x1; presets[2].seq[1]=0x1|0x2;
presets[2].period_ms=500;
    presets[3].len = 3; presets[3].seq[0]=0x2; presets[3].seq[1]=0x4;
presets[3].seq[2]=0x0; presets[3].period_ms=200;
    presets[4].len = 5; presets[4].seq[0]=0x1; presets[4].seq[1]=0x2;
presets[4].seq[2]=0x1; presets[4].seq[3]=0x4; presets[4].seq[4]=0x0;
presets[4].period_ms=150;
}

static void uart_println(const char *s) { USART_DRV_TxStr(s); USART_DRV_TxStr("\n"); }

/* Командный буфер */
#define CMD_BUF_SZ 64

/* обработка готовой команды (строки без \r\n) */
static void process_command(const char *cmd)
```

```
{
    if(strncmp(cmd, "new ", 4) == 0) {
        const char *p = cmd+4;
        char seqchars[8]; int slen = 0;
        while(*p && slen < 8) {
            char c = *p++;
            if(c==' ' || c=='\t') continue;
            if(c=='\r' || c=='\n') break;
            if(c=='g'||c=='r'||c=='y'||c=='n' || c=='G'||c=='R'||c=='Y'||c=='N') {
                seqchars[slen++] = c;
            } else break;
        }
        if(slen < 2) { uart_println("Sequence must be 2..8 symbols (g/r/y/n)"); return;
}
        uart_println("Enter period: 1-fast(200),2-medium(500),3-slow(1000)");
        awaiting_period = true;
        memcpy(pending_seq, seqchars, slen);
        pending_len = slen;
        awaiting_period = true;
    } else if(strncmp(cmd, "set interrupts on", 17) == 0) {
        USART_DRV_EnableInterrupts(true);
        uart_println("interrupts on");
    } else if(strncmp(cmd, "set interrupts off", 18) == 0) {
        USART_DRV_EnableInterrupts(false);
        uart_println("interrupts off");
    } else if(strncmp(cmd, "set ", 4) == 0) {
        int idx = atoi(cmd+4);
        if(idx >=1 && idx <= total_presets) {
            active_seq = idx;
            uart_println("ok");
        } else {
            uart_println("invalid index");
        }
    } else {
        uart_println("unknown command");
    }
}
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */
```

```
  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART6_UART_Init();
  /* USER CODE BEGIN 2 */
  LED_Init();
  BUTTON_Init();
  USART_DRV_Init(&huart6);
  USART_DRV_EnableInterrupts(true);
  USART_DRV_SetEcho(true);
  init_presets_defaults();

  uint32_t last_step_time = HAL_GetTick();
  uart_println("welcome!");
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
       BUTTON_Process();
       if(BUTTON_WasPressed()){
        active_seq++;
        if(active_seq > 4){
              if(active_seq > total_presets) active_seq = 1;
        }
       }

       USART_DRV_PollTx();

       // UART прием
       uint8_t ch;
       while(USART_DRV_PollGetByte(&ch)){
        if(ch=='\n' || ch=='\r'){
              linebuf[linepos]=0;
              if(linepos>0){
                    if(awaiting_period){
                          if(linebuf[0]=='1'||linebuf[0]=='2'||linebuf[0]=='3'){
                                uint32_t period=500;
                                if(linebuf[0]=='1') period=200;
                                if(linebuf[0]=='2') period=500;
                                if(linebuf[0]=='3') period=1000;
                                for(int i=0;i<pending_len;i++)
presets[next_user_slot].seq[i]=char_to_mask(pending_seq[i]);
                                presets[next_user_slot].len = pending_len;
                                presets[next_user_slot].period_ms = period;
                                char tmp[32];
                                snprintf(tmp,sizeof(tmp),"saved %d", next_user_slot);
                                uart_println(tmp);
```

```
                                    if(next_user_slot > total_presets) total_presets =
next_user_slot;

                                    next_user_slot++;
                                    if(next_user_slot > 8) next_user_slot = 5;
                                    awaiting_period=false;
                              } else {
                                    uart_println("invalid period; enter 1/2/3");
                              }
                        } else {
                              process_command(linebuf);
                        }
                  }
                  linepos=0;
             } else {
                  if(linepos<(int)sizeof(linebuf)-1) linebuf[linepos++]=(char)ch;
             }
            }

            // анимация
            uint32_t now = HAL_GetTick();
            if(now - last_step_time >= presets[active_seq].period_ms){
             last_step_time = now;
             uint8_t mask = presets[active_seq].seq[
seq_index[active_seq]%presets[active_seq].len ];
             LED_SetMask(mask);
             seq_index[active_seq] = (seq_index[active_seq]+1)%presets[active_seq].len;
            }
        /* USER CODE END WHILE */

        /* USER CODE BEGIN 3 */
      }
      /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE3);

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
```

```
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSI;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_0) != HAL_OK)
  {
    Error_Handler();
  }
}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

usart_drv.c:

```c
#include "usart_drv.h"
#include "stm32f4xx_hal.h"
#include <string.h>
#include <stdbool.h>

#define UART_TX_BUF_SIZE 256

static UART_HandleTypeDef *huart_ptr = NULL;

/* RX */
static volatile uint8_t rx_ring[UART_RX_BUF_SIZE];
static volatile uint16_t rx_head = 0, rx_tail = 0;

/* TX */
static volatile uint8_t tx_ring[UART_TX_BUF_SIZE];
static volatile uint16_t tx_head = 0, tx_tail = 0;
static volatile bool tx_busy = false;

static volatile bool interrupts_enabled = true;
static bool echo = true;

void USART_DRV_Init(UART_HandleTypeDef *huart)
{
    huart_ptr = huart;
    rx_head = rx_tail = 0;
    tx_head = tx_tail = 0;
    tx_busy = false;
    interrupts_enabled = true;
    __HAL_UART_ENABLE_IT(huart_ptr, UART_IT_RXNE);
}

void USART_DRV_EnableInterrupts(bool en)
{
    if(!huart_ptr) return;
    interrupts_enabled = en;
    if(en) {
        __HAL_UART_ENABLE_IT(huart_ptr, UART_IT_RXNE);
        if(tx_head != tx_tail)
            __HAL_UART_ENABLE_IT(huart_ptr, UART_IT_TXE);
    } else {
        __HAL_UART_DISABLE_IT(huart_ptr, UART_IT_RXNE);
        __HAL_UART_DISABLE_IT(huart_ptr, UART_IT_TXE);
    }
}

bool USART_DRV_IsInterruptsEnabled(void){ return interrupts_enabled; }

bool USART_DRV_TxBuf(const uint8_t *buf, uint16_t len)
{
    if(!huart_ptr) return false;

    uint32_t prim = __get_PRIMASK();
    __disable_irq();

    for(uint16_t i = 0; i < len; i++) {
```

```
        uint16_t next = (tx_head + 1) % UART_TX_BUF_SIZE;
        if(next == tx_tail) {
            // буфер преполнен
            return false;
        }
        tx_ring[tx_head] = buf[i];
        tx_head = next;
    }

    __set_PRIMASK(prim);

    if(interrupts_enabled) {
        if(!tx_busy) {
            tx_busy = true;
            __HAL_UART_ENABLE_IT(huart_ptr, UART_IT_TXE);
        }
    }

    return true;
}

void USART_DRV_PollTx(void)
{
    if(interrupts_enabled || !huart_ptr) return;

    if((tx_head != tx_tail) && (__HAL_UART_GET_FLAG(huart_ptr, UART_FLAG_TXE))) {
        huart_ptr->Instance->DR = tx_ring[tx_tail];
        tx_tail = (tx_tail + 1) % UART_TX_BUF_SIZE;
    }
}

bool USART_DRV_TxStr(const char *s)
{
    return USART_DRV_TxBuf((uint8_t*)s, (uint16_t)strlen(s));
}

bool USART_DRV_PollGetByte(uint8_t *out)
{
    if(USART_DRV_IsInterruptsEnabled()) {
        if(rx_head == rx_tail) return false;

        uint32_t prim = __get_PRIMASK();
        __disable_irq();

        *out = rx_ring[rx_tail++];
        if(rx_tail >= UART_RX_BUF_SIZE) rx_tail = 0;

        __set_PRIMASK(prim);

        if(echo) {
            uint8_t ch = *out;
            USART_DRV_TxBuf(&ch, 1);
        }

        return true;
    } else {
        if(__HAL_UART_GET_FLAG(huart_ptr, UART_FLAG_RXNE)) {
```

```
            *out = (uint8_t)(huart_ptr->Instance->DR & 0xFF);
            if(echo) {
                uint8_t ch = *out;
                USART_DRV_TxBuf(&ch, 1);
            }
            return true;
        }
        return false;
    }
}

void USART_DRV_SetEcho(bool en){ echo = en; }

void USART_DRV_IRQ_Handler(void)
{
    if(!huart_ptr) return;

    uint32_t sr = huart_ptr->Instance->SR;

    /* --- RX --- */
    if(sr & USART_SR_RXNE) {
        uint8_t b = (uint8_t)(huart_ptr->Instance->DR & 0xFF);
        uint16_t next = (rx_head + 1) % UART_RX_BUF_SIZE;
        if(next == rx_tail) {
            // переполнение — дропаем старый байт
            rx_tail = (rx_tail + 1) % UART_RX_BUF_SIZE;
        }
        rx_ring[rx_head] = b;
        rx_head = next;
    }

    /* --- TX --- */
    if((sr & USART_SR_TXE) && tx_head != tx_tail) {
        huart_ptr->Instance->DR = tx_ring[tx_tail];
        tx_tail = (tx_tail + 1) % UART_TX_BUF_SIZE;
    }

    if(tx_head == tx_tail) {
        __HAL_UART_DISABLE_IT(huart_ptr, UART_IT_TXE);
        tx_busy = false;
    }
}
```

Вывод

В ходе выполнения лабораторной работы мы разработали драйвер для управления гирляндой посредством интерфейса UART (с прерываниями и без), обеспечивающий неблокирующий приём и передачу данных с буферизацией для предотвращения потери символов. Программа предоставляет возможность добавления новых режимов гирлянды и переключения между ними.