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

Дисциплина: Проектирование вычислительных систем

Лабораторная работа №3

Вариант 4

Выполнили:
Барсуков Максим Андреевич,
группа P3415
Стригалев Никита Сергеевич,
группа P3412

Преподаватель:
Пинкевич Василий Юрьевич

2025 г.

Санкт-Петербург

Содержание

Задание

Разработать программу, которая использует таймеры для управления яркостью светодиодов и излучателем звука (по прерыванию или с использованием аппаратных каналов). Блокирующее ожидание (функция `HAL_Delay()`) в программе использоваться не должно.

Стенд должен поддерживать связь с компьютером по UART и выполнять указанные действия в качестве реакции на нажатие кнопок на клавиатуре компьютера. В данной лабораторной работе каждая нажатая кнопка (символ, отправленный с компьютера на стенд) обрабатываются отдельно, ожидание ввода полной строки не требуется.

Для работы с UART на стенде можно использован один из двух вариантов драйвера (по прерыванию и по опросу) на выбор исполнителя. Поддержка двух вариантов не требуется.

Частота синхросигнала процессорного ядра и сигнала ШИМ для управления яркостью светодиодов (если используется) должны соответствовать указанным в варианте задания.

Вариант: 4

Реализовать «музыкальную шкатулку» с мелодиями, которые состоят из последовательности звуков определенной частоты и длительности, а также пауз. Шкатулка должна иметь четыре стандартные мелодии и одну пользовательскую, которую можно настроить.

Действия стенда при получении символов от компьютера:

Символ	Действие
«1» – «4»	Воспроизведение одной из стандартных мелодий.
«5»	Воспроизведение пользовательской мелодии.
«Enter»	Вход в меню настройки.
По усмотрению исполнителей	Ввод параметров пользовательской мелодии: частота (нота, октава), длительность, конец мелодии и т. п.

После ввода каждого символа в UART должно выводиться сообщение о том, какой режим активирован, или текущие настройки, вводимые в меню.

Частота процессорного ядра – 180 МГц.

Блок-схемы

Основная программа

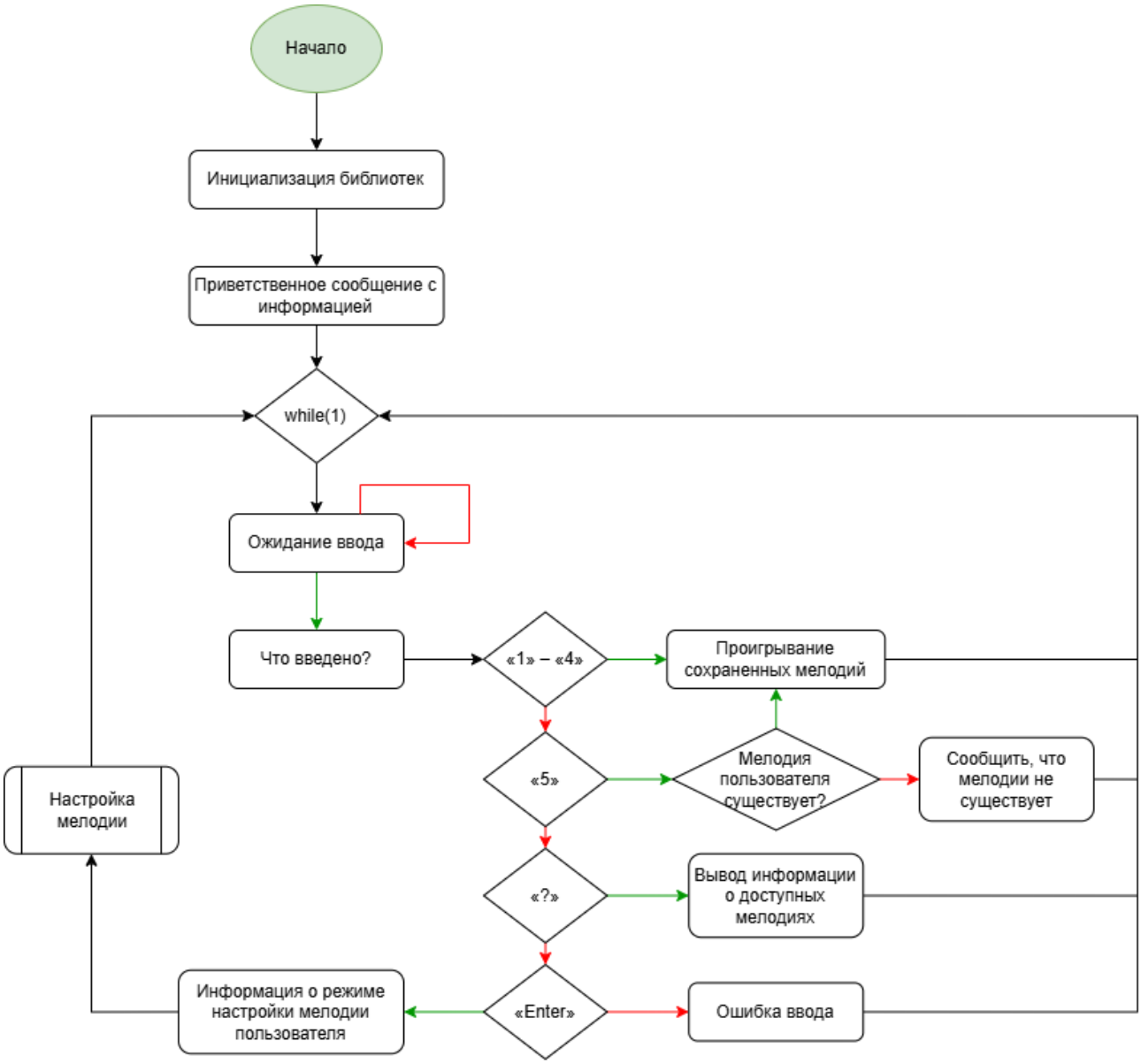

Рисунок 1 – Схема основной программы

Режим настройки мелодии

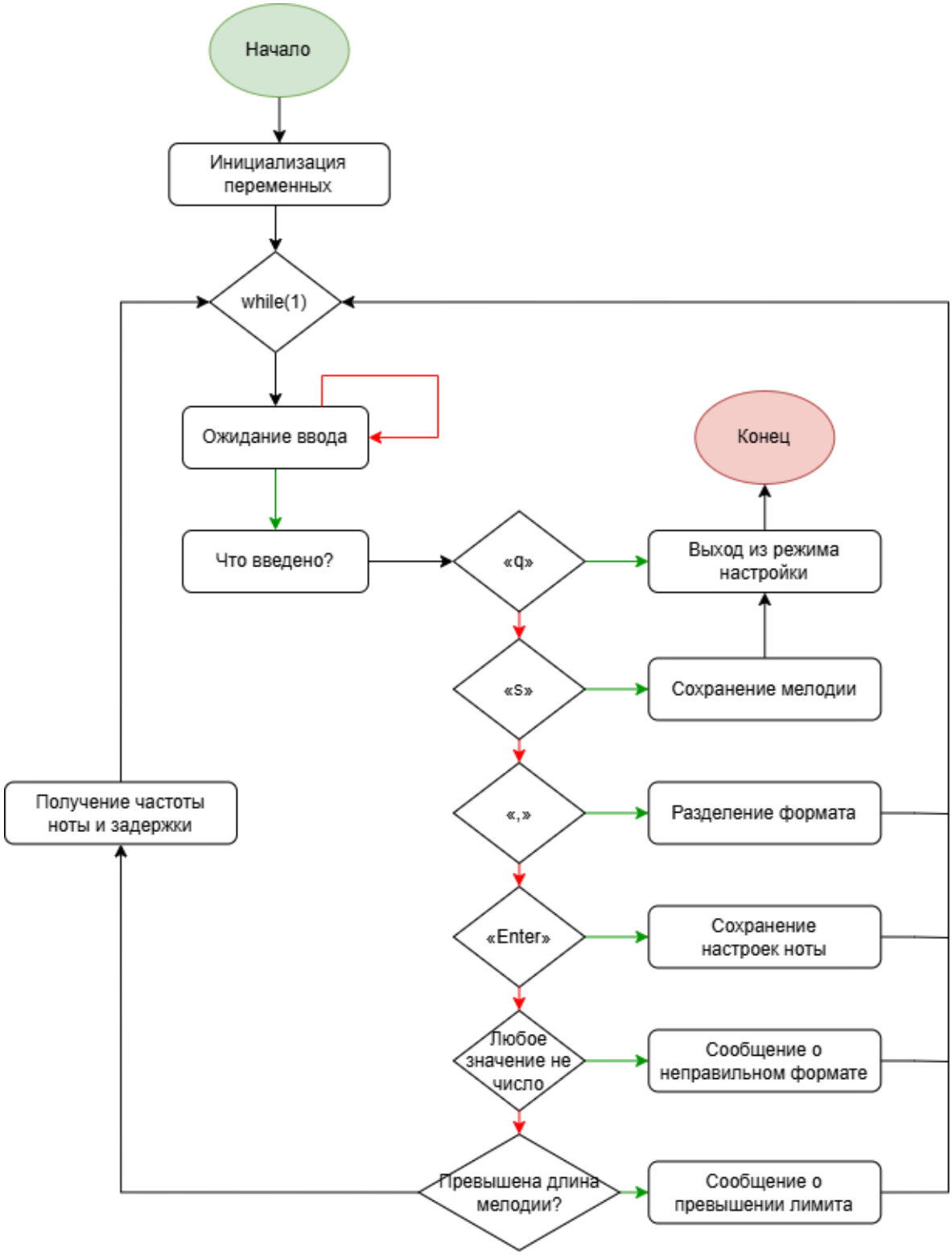

Рисунок 2 – Схема режима настройки мелодии

Исходный код

source.h

```
#ifndef SOURCE_H_
#define SOURCE_H_

#include "stdint.h"

#define C1  33
#define CS1 35
#define D1  37
#define DS1 39
#define E1  41
#define F1  44
#define FS1 46
#define G1  49
#define GS1 52
#define A1  55
#define AS1 58
#define B1  62

#define C2  65
#define CS2 69
#define D2  73
#define DS2 78
#define E2  82
#define F2  87
#define FS2 93
#define G2  98
#define GS2 104
#define A2  110
#define AS2 117
#define B2  123

#define C3  131
#define CS3 139
#define D3  147
#define DS3 156
#define E3  165
#define F3  175
#define FS3 185
#define G3  196
#define GS3 208
#define A3  220
#define AS3 233
#define B3  247

#define C4  262
#define CS4 277
#define D4  294
#define DS4 311
#define E4  330
#define F4  349
```

```
#define FS4 370
#define G4  392
#define GS4 415
#define A4  440
#define AS4 466
#define B4  494

#define C5  523
#define CS5 554
#define D5  587
#define DS5 622
#define E5  659
#define F5  698
#define FS5 740
#define G5  784
#define GS5 831
#define A5  880
#define AS5 932
#define B5  988

#define C6  1047
#define CS6 1109
#define D6  1175
#define DS6 1245
#define E6  1319
#define F6  1397
#define FS6 1480
#define G6  1568
#define GS6 1661
#define A6  1760
#define AS6 1865
#define B6  1976

#define C7  2093
#define CS7 2217
#define D7  2349
#define DS7 2489
#define E7  2637
#define F7  2794
#define FS7 2960
#define G7  3136
#define GS7 3322
#define A7  3520
#define AS7 3729
#define B7  3951

#define C8  4186
#define CS8 4435
#define D8  4699
#define DS8 4978

#define VOLUME_MAX 10
#define VOLUME_MIN 0

extern uint32_t stairway_melody[];
```

```
extern uint32_t stairway_delays[];

extern uint32_t deftones_melody[];
extern uint32_t sssshhhiiiittt_delays[];

extern uint32_t duvet_melody[];
extern uint32_t duvet_delays[];

extern uint32_t radiohead_melody[];
extern uint32_t radiohead_delays[];

extern uint32_t user_melody[256];
extern uint32_t user_delays[256];
extern uint32_t user_melody_size;

// Стартовая мелодия (короткая)
extern uint32_t start_melody[];
extern uint32_t start_delays[];

// Стоп мелодия (тишина)
extern uint32_t stop_melody[];
extern uint32_t stop_delays[];

#endif
```

main.h

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.h
  * @brief          : Header for main.c file.
  *                   This file contains the common defines of the application.
  ******************************************************************************
  * @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 */
/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __MAIN_H
#define __MAIN_H
#ifdef __cplusplus
extern "C" {
#endif
/* Includes ------------------------------------------------------------------*/
#include "stm32f4xx_hal.h"
```

```
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "source.h"
#include <stdbool.h>
/* USER CODE END Includes */
/* Exported types ------------------------------------------------------------*/
/* USER CODE BEGIN ET */
/* USER CODE END ET */
/* Exported constants --------------------------------------------------------*/
/* USER CODE BEGIN EC */
/* USER CODE END EC */
/* Exported macro ------------------------------------------------------------*/
/* USER CODE BEGIN EM */
/* USER CODE END EM */
/* Exported functions prototypes ---------------------------------------------*/
void Error_Handler(void);
/* USER CODE BEGIN EFP */
extern uint32_t custom_track_data[256];
extern uint32_t custom_track_timing[256];
extern uint32_t custom_track_length;
extern uint32_t current_duration;
extern bool is_track_playing;
extern uint32_t track_position;
extern uint32_t *current_track_notes;
extern uint32_t *current_track_durations;
extern uint32_t track_notes_count;
void start_track_playback(uint32_t *notes_array, uint32_t *timings_array, uint32_t
array_length);
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *timer_instance);
/* USER CODE END EFP */
/* Private defines -----------------------------------------------------------*/
/* USER CODE BEGIN Private defines */
/* USER CODE END Private defines */
#ifdef __cplusplus
}
#endif
#endif /* __MAIN_H */
```

main.c

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2022 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 "tim.h"
#include "usart.h"
#include "gpio.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

#include "source.h"
#include <stdio.h>
#include <stdarg.h>
#include <usart.h>
#include <string.h>
#include <ctype.h>
#include <ctype.h>
#include <stdbool.h>


/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define UART_TIMEOUT 10
#define BUF_SIZE 256
#define MAX_DIGITS 5
/* USER CODE END PD */

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

/* USER CODE END PM */

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

/* USER CODE BEGIN PV */
uint32_t custom_track_data[256];
uint32_t custom_track_timing[256];
uint32_t custom_track_length = 0;
uint32_t current_duration = 1;
bool is_track_playing = true;
uint32_t track_position = 0;
uint32_t *current_track_notes;
uint32_t *current_track_durations;
uint32_t track_notes_count;
```

```
uint32_t stairway_melody[] = {
    A2, C3, E3, A4, B4, E3, C3, B4, C4, E3, C2, C4, FS3, D3, A2, FS3
};

uint32_t stairway_delays[] = {
    450, 450, 450, 450, 450, 450, 450, 450, 450, 450, 450, 450, 450, 450, 450, 450,
450
};

uint32_t ssshhhiiittt_melody[] = {
    E4, DS4, CS4, DS4, CS4, 0, B3, 0, CS4, 0, B3, CS4, DS4, 0, B3,
     E4, DS4, CS4, DS4, CS4, 0, B3, 0, CS4, 0, B3, CS4, DS4, 0, AS3
};

uint32_t ssshhhiiittt_delays[] = {
    400, 400, 400, 400, 400, 70, 400, 70, 400, 70, 400, 300, 400, 100, 400,
     400, 400, 400, 400, 400, 70, 400, 70, 400, 70, 400, 400, 400, 100, 400
};

uint32_t duvet_melody[] = {
    D5, CS5, B4, CS5, D5, 0, E5, FS5, B4, 0,
     D5, CS5, B4, CS5, D5, 0, E5, 0, FS5, E5, 0, FS5, E5, 0, E5, D5, CS5
};

uint32_t duvet_delays[] = {
    150 * 3, 150* 3, 150* 3, 150* 3, 150* 3, 50, 150* 3, 150* 3, 250* 3, 50,
    150* 3, 150* 3, 150* 3, 150* 3, 150* 3, 50, 150* 3, 100, 150* 3, 150* 3, 50, 150*
3, 150* 3, 100, 250* 3, 150* 3, 250 * 3
};

uint32_t radiohead_melody[] = {
    FS4, A3, D4, A3, FS4, A3, D4, A3,
    FS4, A3, D4, A3, G3, AS3, D4, E4,
     FS4, A3, D4, A3, FS4, A3, D4, A3,
     FS4, A3, D4, A3, G3, AS3, D4, E4
};

uint32_t radiohead_delays[] = {
    450, 450, 450, 450, 450, 450, 450, 450,
     450, 450, 450, 450, 450, 450, 450, 450,
     450, 450, 450, 450, 450, 450, 450, 450,
     450, 450, 450, 450, 450, 450, 450, 450
};

uint32_t user_melody[256] = {0};
uint32_t user_delays[256] = {0};
uint32_t user_melody_size = 0;

uint32_t start_melody[] = {C5, 0, E5, 0, G5};
uint32_t start_delays[] = {200, 50, 200, 50, 400};

uint32_t stop_melody[] = {0};
uint32_t stop_delays[] = {100};

/* USER CODE END PV */
```

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

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
struct RingBuffer {
    char data[BUF_SIZE];
    uint16_t head;
    uint16_t tail;
    uint16_t count;
};

typedef struct RingBuffer RingBuffer;

static void buf_init(RingBuffer *buf) {
    buf->head = 0;
    buf->tail = 0;
    buf->count = 0;
    memset(buf->data, 0, BUF_SIZE);
}

static bool buf_push(RingBuffer *buf, char c) {
    if (buf->count >= BUF_SIZE) {
        return false;
    }

    buf->data[buf->head] = c;
    buf->head = (buf->head + 1) % BUF_SIZE;
    buf->count++;
    return true;
}

static bool buf_pop(RingBuffer *buf, char *c) {
    if (buf->count == 0) {
        return false;
    }

    *c = buf->data[buf->tail];
    buf->tail = (buf->tail + 1) % BUF_SIZE;
    buf->count--;
    return true;
}

static RingBuffer rx_buffer;

static RingBuffer ringBufferRx;
static RingBuffer ringBufferTx;
static char input_char[2] = {"\0"};
static uint8_t uart_rx_char;

struct SystemState {
```

```
    bool interrupts_active;
};

static struct SystemState system_state;
bool transmit_busy = false;

void activate_interrupts(struct SystemState *state) {
    HAL_NVIC_EnableIRQ(USART6_IRQn);
    state->interrupts_active = true;
    HAL_UART_Receive_IT(&huart6, (uint8_t*)&uart_rx_char, 1);
}

void deactivate_interrupts(struct SystemState *state) {
    HAL_NVIC_DisableIRQ(USART6_IRQn);
    state->interrupts_active = false;
}

void send_uart(char *buffer, size_t length) {
    HAL_UART_Transmit(&huart6, (uint8_t*)buffer, length, 100);
}

void send_uart_line(char *buffer, size_t length) {
    send_uart(buffer, length);
    send_uart("\r\n", 2);
}

void receive_uart_data(const struct SystemState *state) {
    if (state->interrupts_active) {
        return;
    }

    HAL_StatusTypeDef result = HAL_UART_Receive(&huart6, (uint8_t*)input_char, 1, 0);
    if (result == HAL_OK) {
        buf_push(&ringBufferRx, input_char);
        send_uart_data(state, input_char, 1);
    }
}


void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart_instance) {
    if (huart_instance->Instance == USART6) {
        buf_push(&rx_buffer, uart_rx_char);
        send_uart(&uart_rx_char, 1);
        HAL_UART_Receive_IT(&huart6, (uint8_t*)&uart_rx_char, 1);
    }
}

void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart_instance) {
    char output_buffer[1024];
    if (buf_pop(&ringBufferTx, output_buffer)) {
        HAL_UART_Transmit_IT(&huart6, (uint8_t*)output_buffer,
strlen(output_buffer));
    } else {
        transmit_busy = false;
    }
```

```
}

void start_track_playback(uint32_t *notes_array, uint32_t *timings_array, uint32_t
array_length) {
    track_position = 0;
    current_track_notes = notes_array;
    current_track_durations = timings_array;
    track_notes_count = array_length;
    is_track_playing = true;
    current_duration = 1;
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
    if (htim->Instance == TIM6) {
        if (!is_track_playing) {
            return;
        }

        if (current_duration > 0) current_duration--;

        if (current_duration == 0) {
            if (track_position >= track_notes_count) {
                is_track_playing = false;
                __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, 0);
                return;
            }

            uint32_t note_freq = current_track_notes[track_position];
            uint32_t note_duration = current_track_durations[track_position];

            if (note_freq == 0) {
                // Пауза
                __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, 0);
                current_duration = note_duration;
            } else {
                // Нота
                current_duration = note_duration;

                // Расчет для 180 МГц: TIM1 Prescaler = 89 -> тактовая = 2 МГц
                // ARR = 2000000 / freq - 1
                uint32_t arr_value = (2000000 / note_freq) - 1;

                __HAL_TIM_SET_AUTORELOAD(&htim1, arr_value);
                __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, arr_value / 2);
            }
            track_position++;

            if (track_position >= track_notes_count) {
                is_track_playing = false;
                __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, 0);
            }
        }
    }
}
/* USER CODE END 0 */
```

```c
/**
  * @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();
  MX_TIM1_Init();
  MX_TIM6_Init();
  /* USER CODE BEGIN 2 */
  HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
      HAL_TIM_Base_Start_IT(&htim6);

      bool program_initialized = true;
      current_track_notes = start_melody;
      current_track_durations = start_delays;
      track_notes_count = sizeof(start_melody) / sizeof(uint32_t);

      char welcome_message[] = "Добро пожаловать в музыкальную шкатулку!";
      char menu_options[] =
          "\r\nДоступные мелодии:\r\n"
          "\t1. - Led Zeppelin: Stairway to Heaven\r\n"
          "\t2. - Deftones: Be Quiet and Drive\r\n"
          "\t3. - Boa: Duvet\r\n"
          "\t4. - Radiohead: No Surprises\r\n"
          "\t5. - Пользовательская мелодия\r\n"
          "\tEnter - Настройка пользовательской мелодии\r\n"
          "\tx - Остановить воспроизведение";

      char playback_stopped_msg[] = " - Воспроизведение остановлено!";
      char invalid_selection_msg[] = " - Неверный выбор!";
      char stairway_playing_msg[] = " - Играет: \"Stairway to Heaven\"";
      char deftones_playing_msg[] = " - Играет: \"Серф Панк\"";
```

```
      char duvet_playing_msg[] = " - Играет: \"Duvet\"";
      char radiohead_playing_msg[] = " - Играет: \"No Surprises\"";
      char custom_track_playing_msg[] = " - Играет пользовательская мелодия";
      char no_custom_track_msg[] = " - Пользовательская мелодия не создана!";

      char settings_menu[] =
          "Режим настройки пользовательской мелодии!\r\n"
          "Формат ввода: ЧАСТОТА,ДЛИТЕЛЬНОСТЬ\\r\\n\r\n"
          "'q' - выход без сохранения\r\n"
          "'s' - сохранить и выйти";

      char note_added_msg[] = " - Нота добавлена!";
      char format_error_msg[] = " - Ошибка формата! Повторите ввод.";
      char track_too_long_msg[] = "\r\nМелодия слишком длинная!";
      char track_saved_msg[] = " - Мелодия сохранена!";
      char settings_exit_msg[] = " - Выход из режима настройки";

      buf_init(&rx_buffer);
      activate_interrupts(&system_state);

      send_uart_line(welcome_message, strlen(welcome_message));
      program_initialized = false;
      //buf_init(&ringBufferRx);
      //buf_init(&ringBufferTx);
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
      while (1)
      {
          if (program_initialized){
              send_uart_line(welcome_message, strlen(welcome_message));
              program_initialized = false;
          }

          char received_char;

          // Обрабатываем только по одному символу за раз
          if (buf_pop(&rx_buffer, &received_char)) {
              // ДЕБАГ: выводим что получили
              // char debug_msg[32];
              // sprintf(debug_msg, "[DEBUG: 0x%02X '%c']", received_char,
received_char);
              // send_uart_line(debug_msg, strlen(debug_msg));

              switch (received_char) {
                  case '1': {
                      send_uart_line(stairway_playing_msg,
strlen(stairway_playing_msg));
                      start_track_playback(stairway_melody, stairway_delays,
                                           sizeof(stairway_melody) / sizeof(uint32_t));
                      break;
                  }
                  case '2': {
                      send_uart_line(deftones_playing_msg,
strlen(deftones_playing_msg));
```

```
                        start_track_playback(ssshhhiiittt_melody, ssshhhiiittt_delays,
                                             sizeof(ssshhhiiittt_melody) /
sizeof(uint32_t));
                        break;
                    }
                    case '3': {
                        send_uart_line(duvet_playing_msg, strlen(duvet_playing_msg));
                        start_track_playback(duvet_melody, duvet_delays,
                                             sizeof(duvet_melody) / sizeof(uint32_t));
                        break;
                    }
                    case '4': {
                        send_uart_line(radiohead_playing_msg,
strlen(radiohead_playing_msg));
                        start_track_playback(radiohead_melody, radiohead_delays,
                                             sizeof(radiohead_melody) /
sizeof(uint32_t));
                        break;
                    }
                    case '5': {
                        if (custom_track_length != 0){
                            send_uart_line(custom_track_playing_msg,
strlen(custom_track_playing_msg));
                            start_track_playback(custom_track_data,
custom_track_timing, custom_track_length);
                        } else {
                            send_uart_line(no_custom_track_msg,
strlen(no_custom_track_msg));
                        }
                        break;
                    }
                    case '?': {
                        send_uart_line(menu_options, strlen(menu_options));
                        break;
                    }
                    case 'x': {
                        start_track_playback(stop_melody, stop_delays,
                                             sizeof(stop_melody) / sizeof(uint32_t));
                        send_uart_line(playback_stopped_msg,
strlen(playback_stopped_msg));
                        break;
                    }
                    case 'e': {
                        send_uart_line(settings_menu, strlen(settings_menu));
                        uint32_t temp_notes[256];
                        uint32_t temp_timings[256];
                        uint32_t current_size = 0;
                        uint32_t note_value = 0;
                        uint32_t timing_value = 0;
                        bool comma_detected = false;
                        bool settings_mode = true;

                        while (settings_mode) {
                            char input_char;
                            if (buf_pop(&rx_buffer, &input_char)) {
                                // send_uart(&input_char, 1);
```

```
                                if (input_char == 'q') {
                                    send_uart_line(settings_exit_msg,
strlen(settings_exit_msg));
                                    settings_mode = false;
                                }
                                else if (input_char == 's') {
                                    if (current_size > 0) {
                                        send_uart_line(track_saved_msg,
strlen(track_saved_msg));
                                        for (uint32_t i = 0; i < current_size; i++) {
                                            custom_track_data[i] = temp_notes[i];
                                            custom_track_timing[i] = temp_timings[i];
                                        }
                                        custom_track_length = current_size;
                                    } else {
                                        send_uart_line(" - Мелодия пустая! Нечего
сохранять.", 37);
                                    }
                                    settings_mode = false;
                                }
                                else if (input_char == ',') {
                                    if (comma_detected) {
                                        send_uart_line(format_error_msg,
strlen(format_error_msg));
                                        note_value = 0;
                                        timing_value = 0;
                                        comma_detected = false;
                                    } else {
                                        comma_detected = true;
                                    }
                                }
                                else if (input_char == '\r') {
                                    if (current_size < 256) {
                                        temp_notes[current_size] = note_value;
                                        temp_timings[current_size] = timing_value;
                                        current_size++;

                                        char note_msg[64];
                                        if (note_value == 0) {
                                            snprintf(note_msg, sizeof(note_msg), " -
Пауза %d мс добавлена! (нота %d)", timing_value, current_size);
                                        } else {
                                            snprintf(note_msg, sizeof(note_msg), " -
Нота %d Гц, %d мс добавлена! (нота %d)", note_value, timing_value, current_size);
                                        }
                                        send_uart_line(note_msg, strlen(note_msg));

                                        note_value = 0;
                                        timing_value = 0;
                                        comma_detected = false;
                                    } else {
                                        send_uart_line(track_too_long_msg,
strlen(track_too_long_msg));
                                    }
                                }
```

```
                                else if (!isdigit((unsigned char)input_char)) {
                                    send_uart_line(format_error_msg,
strlen(format_error_msg));
                                    note_value = 0;
                                    timing_value = 0;
                                    comma_detected = false;
                                }
                                else if (current_size >= 256) {
                                    send_uart_line(track_too_long_msg,
strlen(track_too_long_msg));
                                }
                                else {
                                    if (comma_detected) {
                                        timing_value = timing_value * 10 + (input_char
- '0');
                                    } else {
                                        note_value = note_value * 10 + (input_char -
'0');
                                    }
                                }
                            }
                            HAL_Delay(10);
                        }
                        break;
                    }
                    case '\r': // Enter
                    case '\n': // Новая строка
                        break;

                    default: {
                        if (received_char >= 32 && received_char <= 126) {
                            send_uart_line(invalid_selection_msg,
strlen(invalid_selection_msg));
                        }
                        break;
                    }
                }
            }

            // Небольшая задержка для стабильности
            HAL_Delay(50);
        }
    /* 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_SCALE1);

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 15;
  RCC_OscInitStruct.PLL.PLLN = 216;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 4;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Activate the Over-Drive mode
  */
  if (HAL_PWREx_EnableOverDrive() != 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_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != 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 */
```

Вывод

В ходе выполнения лабораторной работы была реализована простая музыкальная шкатулка, способная воспроизводить четыре предопределенные мелодии, а также мелодию, заданную пользователем. Для этого был добавлен специальный режим настройки пользовательской мелодии. Для обеспечения взаимодействия с пользователем через последовательный интерфейс была интегрирована часть кода из предыдущей лабораторной работы, реализующая работу с UART — включая прием и обработку входных данных.