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

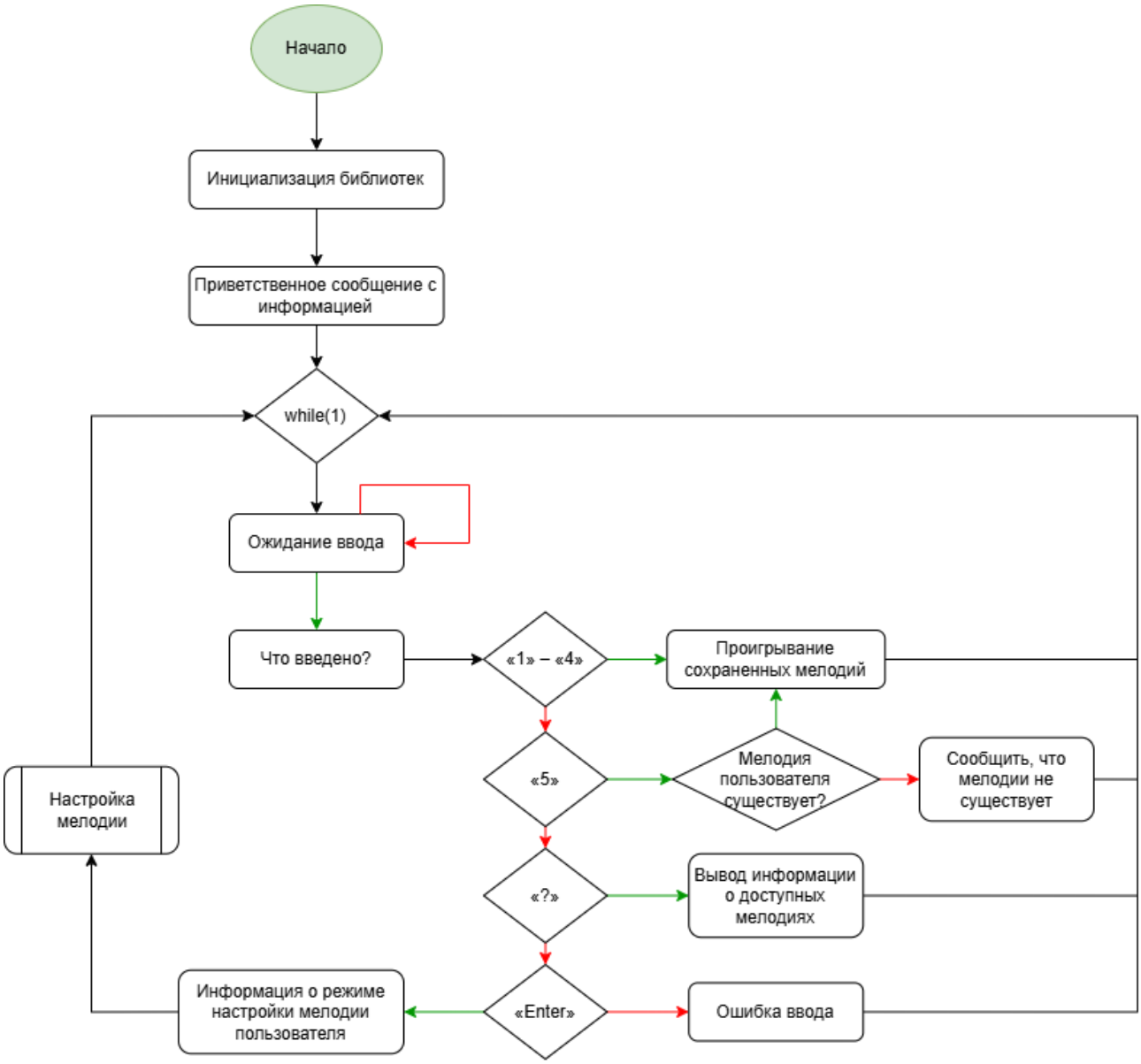

Рисунок 1 – Схема основной программы

Режим настройки мелодии

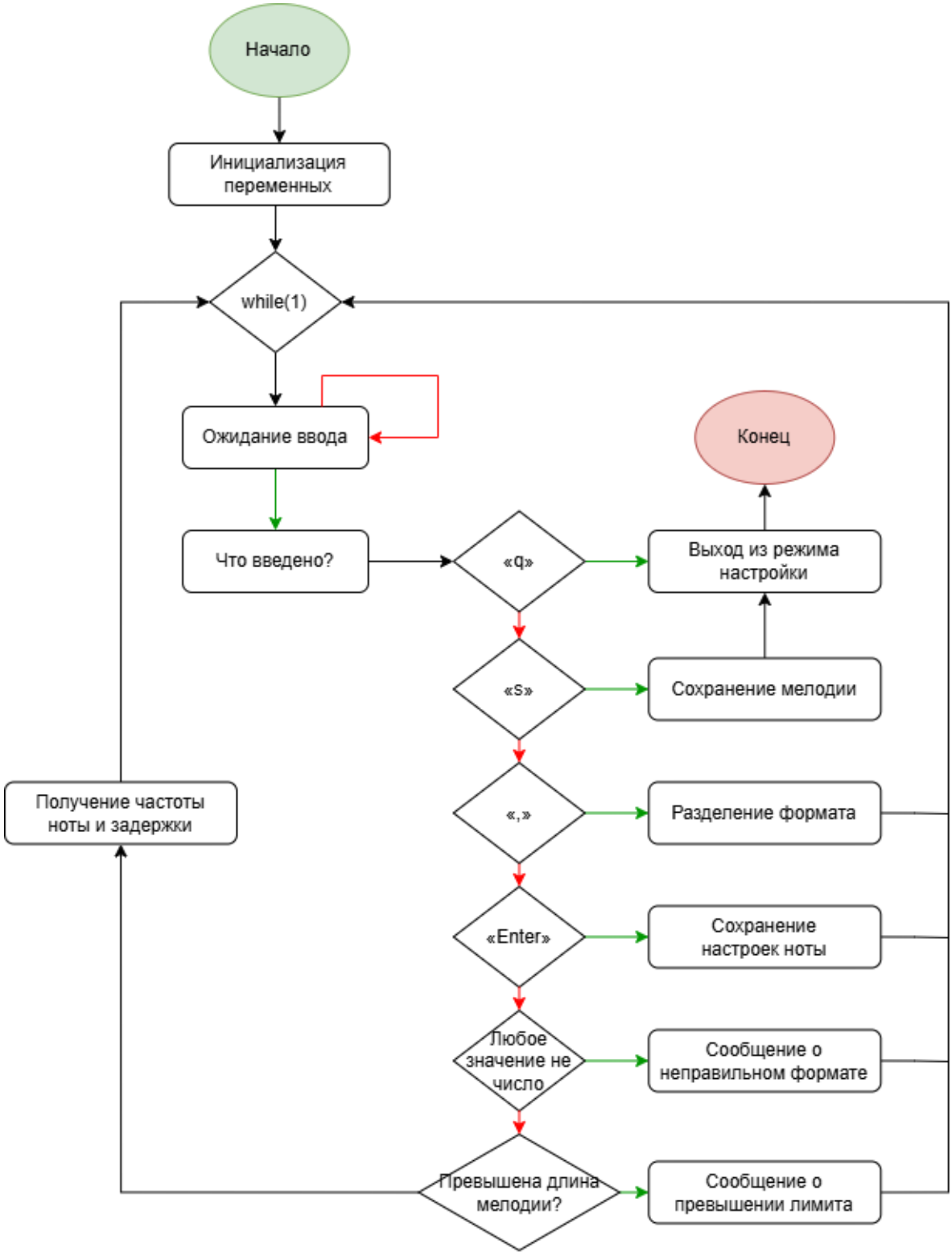

Рисунок 2 – Схема режима настройки мелодии

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

// 1. Stairway to Heaven (Led Zeppelin) - вступление (ПЕРЕПРОВЕРЕНО)
```

```
uint32_t stairway_melody[] = {
    A4, 0, C5, D5, 0, F5, E5, 0, D5, C5, A4,
    G4, 0, A4, C5, 0, D5, E5, D5, C5, A4,
    G4, 0, A4, C5, 0, D5, E5, F5, E5, D5,
    C5, A4, G4, F4, E4, D4, C4, A3,

    A4, 0, C5, D5, 0, F5, E5, 0, D5, C5, A4,
    G4, 0, A4, C5, 0, D5, E5, D5, C5, A4,
    F4, G4, A4, C5, D5, E5, F5, G5,
    A5, 0, G5, F5, E5, D5, C5, A4
};

uint32_t stairway_delays[] = {
    400, 50, 400, 400, 50, 400, 400, 50, 400, 400, 400,
    400, 50, 400, 400, 50, 400, 400, 400, 400, 400,
    400, 50, 400, 400, 50, 400, 400, 400, 400, 400,
    400, 400, 400, 400, 400, 400, 400, 800,

    400, 50, 400, 400, 50, 400, 400, 50, 400, 400, 400,
    400, 50, 400, 400, 50, 400, 400, 400, 400, 400,
    300, 300, 300, 300, 300, 300, 300, 300,
    600, 50, 300, 300, 300, 300, 300, 600
};

// 2. Deftones - Be Quiet And Drive (Far Away)
uint32_t deftones_melody[] = {
    D4, A3, 0, GS3, G3, F3, D3, F3, G3,
    C3, 0, 0, D4, A3, 0, GS3, G3, F3, D3, F3, G3,
    B2, 0, 0, D4, A3, 0, GS3, G3, F3, D3, F3, G3,
    AS2, 0, 0, D4, A3, 0, GS3, G3, F3, D3, F3, G3
};

uint32_t deftones_delays[] = {
    450, 450, 50, 300, 300, 300, 300, 300, 450,
    450, 50, 50, 450, 450, 50, 300, 300, 300, 300, 300, 450,
    450, 50, 50, 450, 450, 50, 300, 300, 300, 300, 300, 450,
    450, 50, 50, 450, 450, 50, 300, 300, 300, 300, 300, 800
};

// 3. Boa - Duvet
uint32_t duvet_melody[] = {
    C5, B4, A4, G4, 0, F4, E4, D4, C4, 0,
    G4, A4, B4, 0, C5, B4, A4, G4, 0,
    F4, E4, D4, C4, 0, G4, A4, B4, 0,
    C5, B4, A4, G4, F4, E4, D4, C4
};

uint32_t duvet_delays[] = {
    350, 350, 350, 350, 50, 350, 350, 350, 350, 50,
    350, 350, 350, 50, 350, 350, 350, 350, 50,
    350, 350, 350, 350, 50, 350, 350, 350, 50,
    350, 350, 350, 350, 350, 350, 350, 800
};

// 4. Radiohead - No Surprises
```

```
uint32_t radiohead_melody[] = {
    E4, G4, C5, G4, E4, G4, C5, G4,
    D4, F4, AS4, F4, D4, F4, AS4, F4,
    E4, G4, C5, G4, E4, G4, C5, G4,
    D4, F4, AS4, F4, D4, F4, AS4, F4,
    C4, E4, G4, E4, C4, E4, G4, E4,
    D4, F4, AS4, F4, D4, F4, AS4, F4
};

uint32_t radiohead_delays[] = {
    300, 300, 300, 300, 300, 300, 300, 300,
    300, 300, 300, 300, 300, 300, 300, 300,
    300, 300, 300, 300, 300, 300, 300, 300,
    300, 300, 300, 300, 300, 300, 300, 300,
    300, 300, 300, 300, 300, 300, 300, 300,
    300, 300, 300, 300, 300, 300, 300, 600
};

// 5. Пользовательская мелодия (пустая)
uint32_t user_melody[64] = {0};
uint32_t user_delays[64] = {0};
uint32_t user_melody_size = 0;

// Стартовая мелодия (короткая)
uint32_t start_melody[] = {C5, 0, E5, 0, G5};
uint32_t start_delays[] = {200, 50, 200, 50, 400};

// Стоп мелодия (тишина)
uint32_t stop_melody[] = {0};
uint32_t stop_delays[] = {100};

#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 */
```

```c
/* 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 */

/* USER CODE END EFP */

/* Private defines -----------------------------------------------------------*/
/* USER CODE BEGIN Private defines */
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 = start_melody;
uint32_t *current_track_durations = start_delays;
uint32_t track_notes_count = sizeof(start_melody) / sizeof(uint32_t);

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;
}

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

        if (current_duration > 0) current_duration--;

        if (current_duration == 0) {
            if (current_track_notes[track_position] == 0) {
                TIM1->CCR1 = 0;
                current_duration = current_track_durations[track_position];
            } else {
                current_duration = current_track_durations[track_position];
                uint32_t prescaler_value = TIM1->PSC;
                uint32_t frequency_value = current_track_notes[track_position];
                TIM1->ARR = 90000000 / (frequency_value * prescaler_value) - 1;
                TIM1->CCR1 = TIM1->ARR >> 1;
            }

            track_position++;

            if (track_position == track_notes_count){
                track_position = 0;
                current_duration = 1;
                is_track_playing = false;
                TIM1->CCR1 = 0;
            }
        }
    }
}

/* 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 1024
#define MAX_DIGITS 5
/* USER CODE END PD */

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

/* USER CODE END PM */

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

/* USER CODE BEGIN PV */
```

```c
/* 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 */
struct CircularQueue {
  char data[BUF_SIZE];
  uint16_t write_pos;
  uint16_t read_pos;
  bool empty;
};

struct ButtonStatus {
    bool pressed;
    bool acknowledged;
    uint32_t pressed_timestamp;
};

struct SystemState {
    bool interrupts_active;
    uint32_t interrupt_mask;
};

typedef struct CircularQueue CircularQueue;
static struct CircularQueue input_queue;
static struct CircularQueue output_queue;
static struct SystemState system_state;

static void queue_init(CircularQueue *queue) {
  queue->write_pos = 0;
  queue->read_pos = 0;
  queue->empty = true;
}

static void queue_push(CircularQueue *queue, char *element) {
  uint16_t element_size = strlen(element);

  if (queue->write_pos + element_size + 1 > BUF_SIZE) {
    queue->write_pos = 0;
  }

  strcpy(&queue->data[queue->write_pos], element);
  queue->write_pos += element_size + 1;

  if (queue->write_pos == BUF_SIZE) {
    queue->write_pos = 0;
  }

  queue->empty = false;
}
```

```
static bool queue_pop(CircularQueue *queue, char *element) {
  if (queue->empty) {
    return false;
  }

  uint16_t element_size = strlen(&queue->data[queue->read_pos]);

  strcpy(element, &queue->data[queue->read_pos]);
  queue->read_pos += element_size + 1;

  if (queue->read_pos == BUF_SIZE || queue->data[queue->read_pos] == '\0') {
    queue->read_pos = 0;
  }

  if (queue->read_pos == queue->write_pos) {
    queue->empty = true;
  }

  return true;
}

static char input_char[2] = {"\0"};

static bool check_button_pressed() {
    return HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_15) == GPIO_PIN_RESET;
}

static void control_led(GPIO_TypeDef* port, uint16_t pin, bool state) {
    HAL_GPIO_WritePin(port, pin, state ? GPIO_PIN_SET : GPIO_PIN_RESET);
}

static bool process_button_status(struct ButtonStatus *status) {
    if (status->pressed) {
        status->pressed = check_button_pressed();

        if (status->acknowledged) {
            return false;
        }

        if ((HAL_GetTick() - status->pressed_timestamp) > 20) {
            status->acknowledged = true;
            return true;
        }
        return false;
    }

    if (check_button_pressed()) {
        status->pressed_timestamp = HAL_GetTick();
        status->pressed = true;
        status->acknowledged = false;
    }

    return false;
}
```

```
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*)input_char, 1);
}

void deactivate_interrupts(struct SystemState *state) {
    HAL_UART_AbortReceive(&huart6);
    HAL_NVIC_DisableIRQ(USART6_IRQn);
    state->interrupts_active = false;
}

void send_uart_data(const struct SystemState *state, char *buffer, size_t length) {
    if (state->interrupts_active) {
        if (transmit_busy) {
            queue_push(&output_queue, buffer);
        } else {
            HAL_UART_Transmit_IT(&huart6, (uint8_t*)buffer, length);
            transmit_busy = true;
        }
        return;
    }
    HAL_UART_Transmit(&huart6, (uint8_t*)buffer, length, 100);
}

void send_uart_line(const struct SystemState *state, char *buffer, size_t length) {
    send_uart_data(state, buffer, length);
    send_uart_data(state, "\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) {
        queue_push(&input_queue, input_char);
        send_uart_data(state, input_char, 1);
    }
}

void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart_instance) {
    queue_push(&input_queue, input_char);
    send_uart_data(&system_state, input_char, 1);
    HAL_UART_Receive_IT(&huart6, (uint8_t*)input_char, 1);
}

void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart_instance) {
    char output_buffer[1024];
    if (queue_pop(&output_queue, output_buffer)) {
        HAL_UART_Transmit_IT(&huart6, (uint8_t*)output_buffer,
```

```
strlen(output_buffer));
    } else {
        transmit_busy = false;
    }
}

enum CalculationStage {
    FirstOperandInput,
    FirstOperandComplete,
    SecondOperandInput,
    SecondOperandComplete,
    ComputeResult,
    ErrorOccurred
};
/* 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();
  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;

      char welcome_message[] = {
          "Добро пожаловать в музыкальную шкатулку!\r\n"
      };
```

```
        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: \"Be Quiet and Drive\""};
        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"};

        activate_interrupts(&system_state);
        queue_init(&input_queue);
        queue_init(&output_queue);
    /* USER CODE END 2 */

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

              receive_uart_data(&system_state);

              char received_symbol[2];

              if (!queue_pop(&input_queue, received_symbol)) {
                  continue;
              }
```

```
            switch (received_symbol[0]) {
                case '1': {
                    send_uart_line(&system_state, stairway_playing_msg,
sizeof(stairway_playing_msg));
                    start_track_playback(stairway_melody, stairway_delays,
                                         sizeof(stairway_melody) / sizeof(uint32_t));
                    break;
                }
                case '2': {
                    send_uart_line(&system_state, deftones_playing_msg,
sizeof(deftones_playing_msg));
                    start_track_playback(deftones_melody, deftones_delays,
                                         sizeof(deftones_melody) / sizeof(uint32_t));
                    break;
                }
                case '3': {
                    send_uart_line(&system_state, duvet_playing_msg,
sizeof(duvet_playing_msg));
                    start_track_playback(duvet_melody, duvet_delays,
                                         sizeof(duvet_melody) / sizeof(uint32_t));
                    break;
                }
                case '4': {
                    send_uart_line(&system_state, radiohead_playing_msg,
sizeof(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(&system_state, custom_track_playing_msg,
sizeof(custom_track_playing_msg));
                        start_track_playback(custom_track_data, custom_track_timing,
                                             sizeof(custom_track_data) /
sizeof(uint32_t));
                        break;
                    }
                    send_uart_line(&system_state, no_custom_track_msg,
sizeof(no_custom_track_msg));
                    break;
                }
                case '?': {
                    send_uart_line(&system_state, menu_options,
sizeof(menu_options));
                    break;
                }
                case 'x': {
                    start_track_playback(stop_melody, stop_delays,
                                         sizeof(stop_melody) / sizeof(uint32_t));
                    send_uart_line(&system_state, playback_stopped_msg,
sizeof(playback_stopped_msg));
                    break;
                }
                case '\r': {
```

```
                    send_uart_line(&system_state, settings_menu,
sizeof(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;

                    while (1) {
                        receive_uart_data(&system_state);
                        char input_data[2];
                        while (!queue_pop(&input_queue, input_data));

                        if (input_data[0] == 'q') {
                            send_uart_line(&system_state, settings_exit_msg,
sizeof(settings_exit_msg));
                            break;
                        }
                        if (input_data[0] == 's') {
                            send_uart_line(&system_state, track_saved_msg,
sizeof(track_saved_msg));
                            memcpy(custom_track_data, temp_notes,
sizeof(custom_track_data));
                            memcpy(custom_track_timing, temp_timings,
sizeof(custom_track_timing));
                            custom_track_length = sizeof(custom_track_data) /
sizeof(uint32_t);
                            break;
                        }
                        if (input_data[0] == ',') {
                            if (comma_detected) {
                                send_uart_line(&system_state, format_error_msg,
sizeof(format_error_msg));
                                note_value = 0;
                                timing_value = 0;
                                comma_detected = false;
                                continue;
                            }
                            comma_detected = true;
                            continue;
                        }
                        if (input_data[0] == '\r') {
                            temp_notes[current_size] = note_value;
                            temp_timings[current_size] = timing_value;
                            current_size++;
                            note_value = 0;
                            timing_value = 0;
                            comma_detected = false;
                            continue;
                        }
                        if (!isdigit(input_data[0])) {
                            send_uart_line(&system_state, format_error_msg,
sizeof(format_error_msg));
                            note_value = 0;
                            timing_value = 0;
```

```
                                comma_detected = false;
                                continue;
                            }
                            if (current_size == 256) {
                                send_uart_line(&system_state, track_too_long_msg,
sizeof(track_too_long_msg));
                            }
                            if (comma_detected) {
                                timing_value = timing_value * 10 + (input_data[0] - '0');
                            } else {
                                note_value = note_value * 10 + (input_data[0] - '0');
                            }
                        }
                        break;
                    }
                    default: {
                        send_uart_line(&system_state, invalid_selection_msg,
sizeof(invalid_selection_msg));
                        break;
                    }
                }

    /* 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 */
```

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.