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

Дисциплина: Проектирование вычислительных систем

Лабораторная работа №4

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 интерфейс I2C для того, чтобы корректно считывать нажатие кнопок клавиатуры стенда SDK-1.1. Подсистема опроса клавиатуры должна удовлетворять следующим требованиям:

- реализуется защита от дребезга;
- нажатие кнопки фиксируется сразу после того, как было обнаружено, что кнопка нажата (с учетом защиты от дребезга), а не в момент отпускания кнопки; если необходимо, долгое нажатие может фиксироваться отдельно;
- кнопка, которая удерживается дольше, чем один цикл опроса, не считается повторно нажатой до тех пор, пока не будет отпущена (нет переповторов);
- распознается и корректно обрабатывается множественное нажатие (при нажатии более чем одной кнопки считается, что ни одна кнопка не нажата, если это не противоречит требованиям к программе);
- всем кнопкам назначаются коды от 1 до 12 (порядок на усмотрение исполнителей).

Программа должна иметь два режима работы, переключение между которыми производится по нажатию кнопки на боковой панели стенда:

- режим тестирования клавиатуры;
- прикладной режим.

Уведомление о смене режима выводится в UART.

1. В режиме тестирования клавиатуры программа выводит в UART коды нажатых кнопок.
2. В прикладном режиме программа обрабатывает нажатия кнопок и выполняет действия в соответствии с вариантом задания.

Вариант: 4

Задания аналогичны вариантам лабораторной работы №3, за исключением того, что ввод символов должен выполняться не с клавиатуры через UART, а с помощью клавиатуры стенда. Выбор кнопок клавиатуры стенда, играющих роль кнопок клавиатуры компьютера должен выполняться по усмотрению исполнителей. В отчете необходимо привести описание функций кнопок в реализованной программе.

Используемые контакты

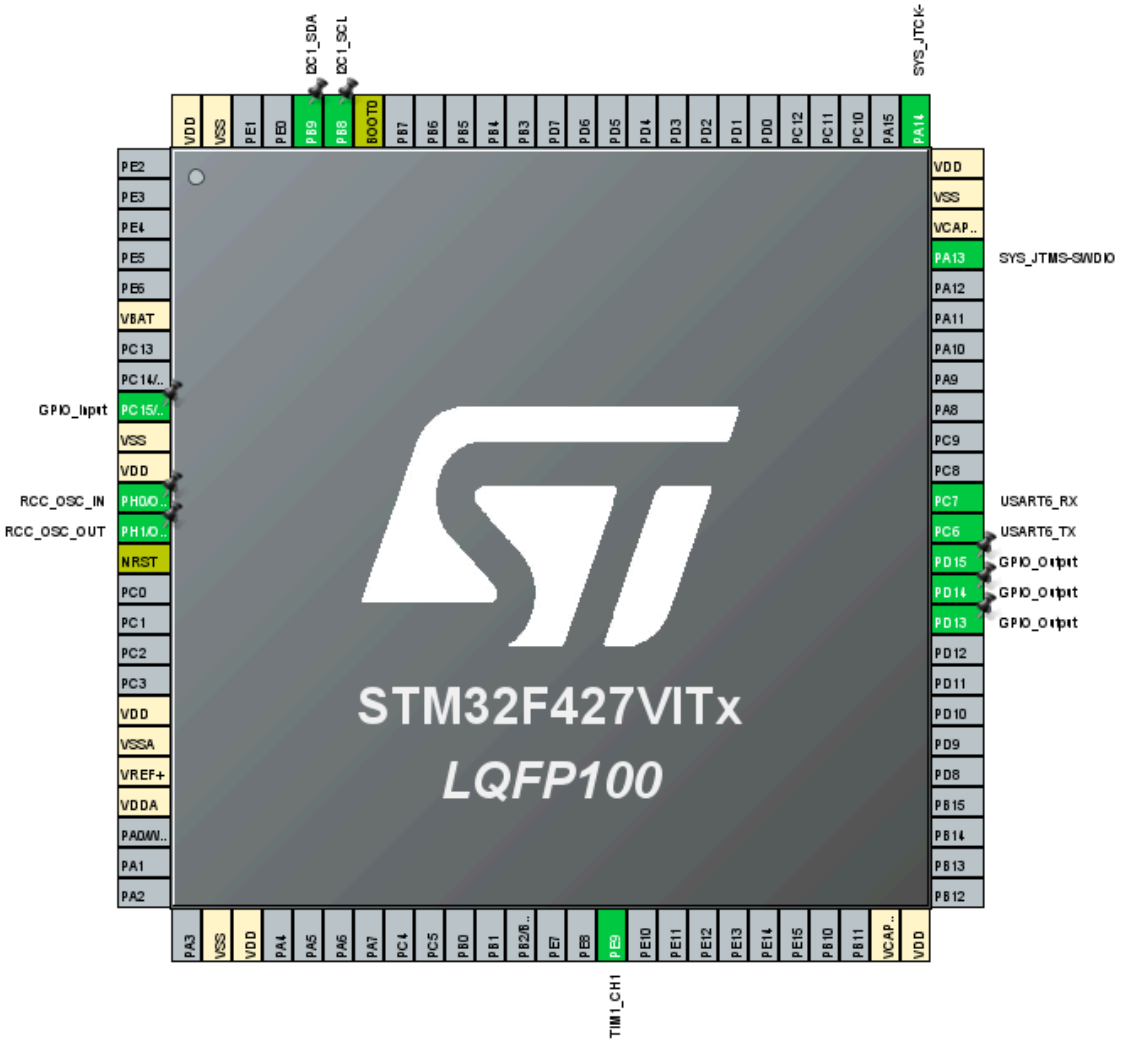

Рисунок 1 – Используемые контакты

Блок-схемы

Основная программа

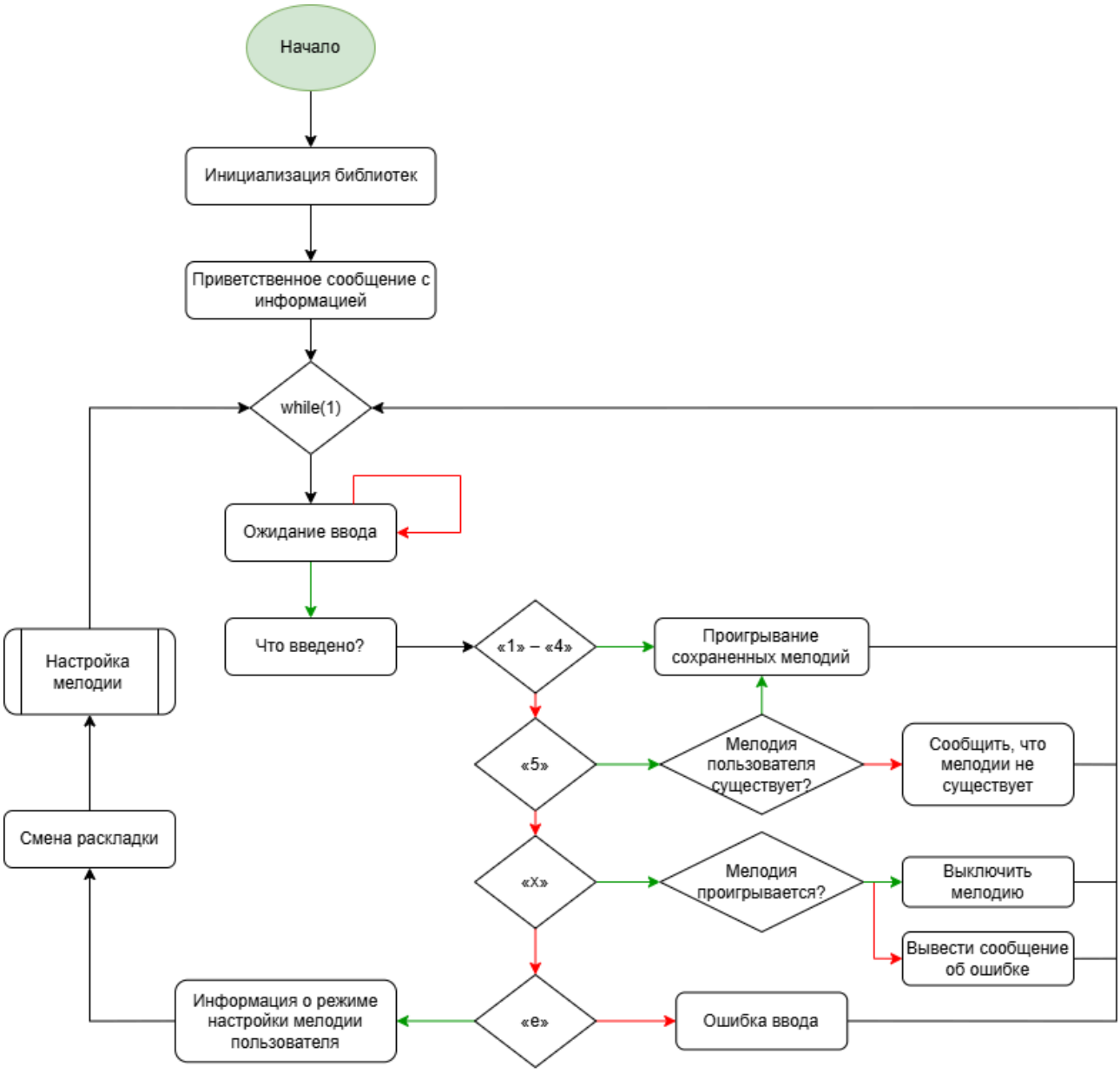

Рисунок 2 – Схема основной программы

Режим настройки мелодии

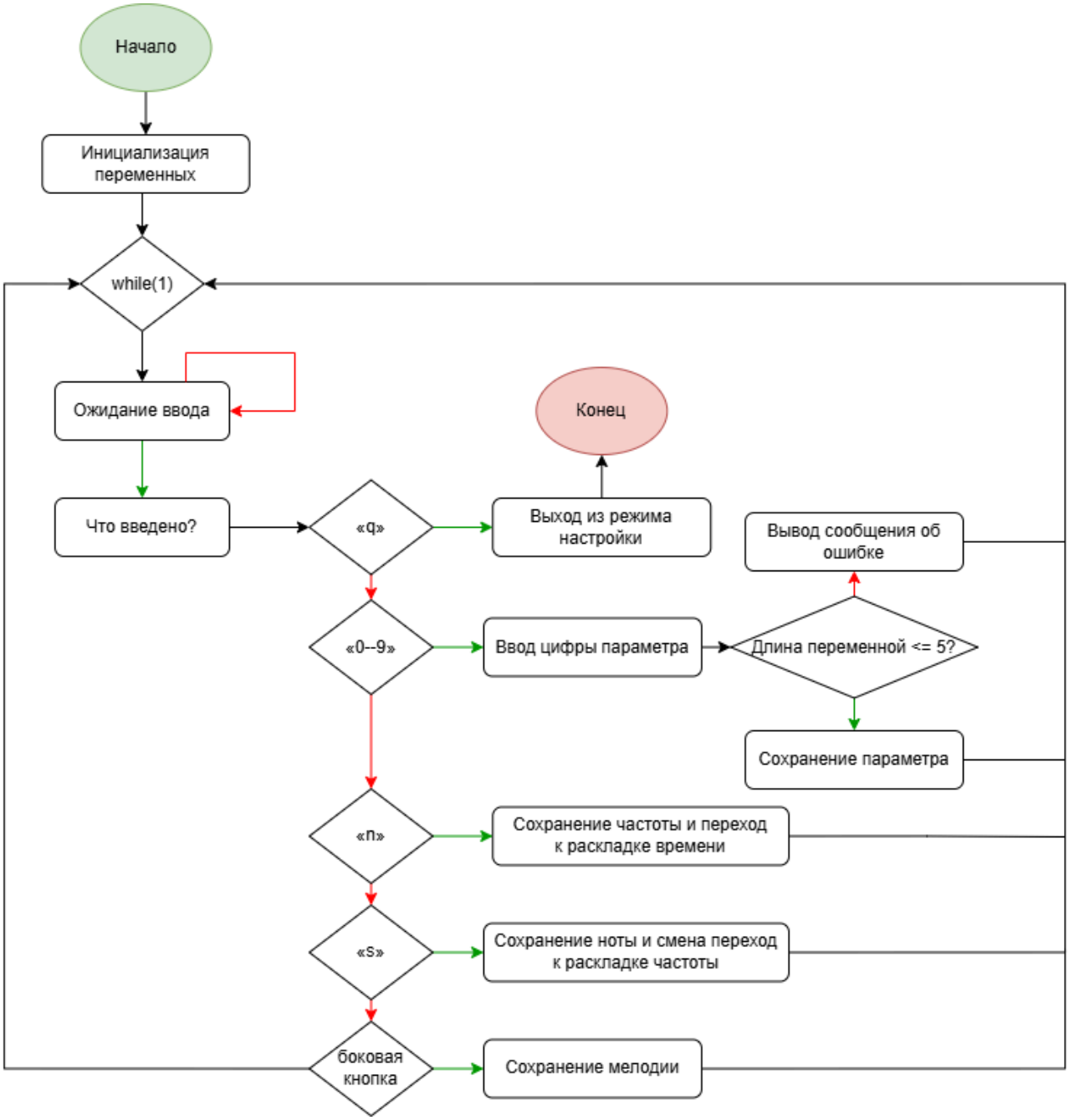

Рисунок 3 – Схема режима настройки мелодии

Описание функций кнопок

Символ	Действие
«1» – «4»	Воспроизведение одной из стандартных мелодий.
«5»	Воспроизведение пользовательской мелодии.
«e»	Вход в меню настройки.
«x»	Остановить мелодию
«q»	Выйти без сохранения из настройки пользовательской мелодии.
«s»	Сохранение ноты.
«n»	Сохранение частоты.
Боковая кнопка	Сохранение мелодии.

Раскладки

Музыкальная раскладка

1	2	3
4	5	
	e	x

Раскладка настройки частоты

1	2	3
4	5	6
7	8	9
0	n	q

Раскладка настройки времени

1	2	3
4	5	6
7	8	9
0	s	q

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 ssshhhiiittt_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;

extern uint32_t start_melody[];
extern uint32_t start_delays[];

extern uint32_t stop_melody[];
extern uint32_t stop_delays[];

#endif
```

main.c

```
/* USER CODE BEGIN PTD */
/* 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 "i2c.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 END PTD */

/* Private define
------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define UART_TIMEOUT 10
#define BUF_SIZE 256
#define MAX_DIGITS 5
#define KB_I2C_ADDRESS (0xE2)
#define KB_I2C_READ_ADDRESS ((KB_I2C_ADDRESS) | 1)
#define KB_I2C_WRITE_ADDRESS ((KB_I2C_ADDRESS) & ~1)
#define KB_INPUT_REG (0x0)
#define KB_OUTPUT_REG (0x1)
#define KB_CONFIG_REG (0x3)
#define KB_KEY_DEBOUNCE_TIME (200)

/* 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 note_value = 0;
uint32_t timing_value = 0;
bool comma_detected = false;

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 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};

uint32_t last_pressing_time = 0;
int last_pressed_btn_index = -1;

typedef enum {
    MODE_MUSIC,
    MODE_EDIT_FREQ,
    MODE_EDIT_TIME,
    MODE_TEST
} SystemMode;
```

```
SystemMode current_mode = MODE_MUSIC;
SystemMode previous_mode = MODE_MUSIC;

bool is_music_mode = true;
bool is_settings_mode = false;
bool is_test_keyboard_mode = false;
bool last_btn_state = false;

char music_layout[] =      {'3', '2', '1',
                            '6', '5', '4',
                            '9', '8', '7',
                            'x', 'e', '0'};

char edit_freq_layout[] =  {'3', '2', '1',
                            '6', '5', '4',
                            '9', '8', '7',
                            'q', 'n', '0'};

char edit_time_layout[] =  {'3', '2', '1',
                            '6', '5', '4',
                            '9', '8', '7',
                            'q', 's', '0'};

char test_layout[] =       {'3', '2', '1',
                            '6', '5', '4',
                            '9', '8', '7',
                            '!', 't', '0'};



uint32_t temp_notes[256];
uint32_t temp_timings[256];
uint32_t temp_size = 0;
uint32_t current_note_freq = 0;
uint32_t current_note_time = 0;
bool is_editing_freq = true;

char freq_buffer[32] = {0};
char time_buffer[32] = {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 void set_green_led(bool on) {
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, on ? GPIO_PIN_SET : GPIO_PIN_RESET);
}

static void set_yellow_led(bool on) {
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, on ? GPIO_PIN_SET : GPIO_PIN_RESET);
}

static void set_red_led(bool on) {
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, on ? GPIO_PIN_SET : GPIO_PIN_RESET);
}

void blink_leds(int count) {
    for (int i = 0; i < count; i++) {
        set_green_led(true);
        set_yellow_led(true);
        set_red_led(true);
        HAL_Delay(100);
        set_green_led(false);
        set_yellow_led(false);
        set_red_led(false);
        if (i < count - 1) HAL_Delay(100);
    }
}

bool is_mode_btn_pressed(void) {
    return HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_15) == GPIO_PIN_RESET;
}

int get_pressed_btn_index(void) {
    const uint32_t t = HAL_GetTick();

    if (t - last_pressing_time < KB_KEY_DEBOUNCE_TIME) {
        return -1;
    }

    int index = -1;
    uint8_t reg_buffer = 0xFF;
    uint8_t tmp = 0;
    int detected_keys = 0;
    int row_values[4] = {0};
    int col_values[3] = {0};

    HAL_I2C_Mem_Write(&hi2c1, KB_I2C_WRITE_ADDRESS, KB_OUTPUT_REG, 1, &tmp, 1,
KB_KEY_DEBOUNCE_TIME);

    for (int row = 0; row < 4; row++) {
```

```
        uint8_t config_value = ~((uint8_t)(1 << row));
        HAL_I2C_Mem_Write(&hi2c1, KB_I2C_WRITE_ADDRESS, KB_CONFIG_REG, 1,
&config_value, 1, KB_KEY_DEBOUNCE_TIME);

        HAL_Delay(2);

        HAL_I2C_Mem_Read(&hi2c1, KB_I2C_READ_ADDRESS, KB_INPUT_REG, 1,
&reg_buffer, 1, KB_KEY_DEBOUNCE_TIME);

        uint8_t column_state = (reg_buffer >> 4) & 0x07;

        if ((column_state & 0x04) == 0) {
            if (detected_keys == 0) index = row * 3 + 0;
            detected_keys++;
            row_values[row]++;
            col_values[0]++;
        }
        if ((column_state & 0x02) == 0) {
            if (detected_keys == 0) index = row * 3 + 1;
            detected_keys++;
            row_values[row]++;
            col_values[1]++;
        }
        if ((column_state & 0x01) == 0) {
            if (detected_keys == 0) index = row * 3 + 2;
            detected_keys++;
            row_values[row]++;
            col_values[2]++;
        }
    }

    if (detected_keys > 2) {
        return -1;
    }

    int rows_pressed = 0;
    int cols_pressed = 0;
    for (int i = 0; i < 4; i++) {
        if (row_values[i] > 0) rows_pressed++;
    }
    for (int i = 0; i < 3; i++) {
        if (col_values[i] > 0) cols_pressed++;
    }

    if (rows_pressed != 1 || cols_pressed != 1) {
        return -1;
    }

    if (detected_keys == 1 && index != -1) {
        if (index == last_pressed_btn_index) {
            return -1;
```

```
            }
            last_pressed_btn_index = index;
            last_pressing_time = t;
            return index;
        }

        if (detected_keys == 0) {
            last_pressed_btn_index = -1;
        }

        return -1;
    }

    char key2char(int key_index) {
        if (key_index < 0 || key_index >= 12) {
            return '!';
        }

        switch (current_mode) {
            case MODE_EDIT_FREQ:
                return edit_freq_layout[key_index];
            case MODE_EDIT_TIME:
                return edit_time_layout[key_index];
            case MODE_TEST:
                return test_layout[key_index];
            case MODE_MUSIC:
            default:
                return music_layout[key_index];
        }
    }

    void enter_edit_mode(void) {
        previous_mode = current_mode;
        current_mode = MODE_EDIT_FREQ;
        is_editing_freq = true;
        current_note_freq = 0;
        current_note_time = 0;

        set_green_led(true);
        set_yellow_led(false);
        set_red_led(false);

        char msg[] = "\r\n=== РЕЖИМ РЕДАКТИРОВАНИЯ ===\r\n"
                     "Редактирование частоты ноты\r\n"
                     "Используйте цифры 0-9 для ввода\r\n"
                     "'n' - перейти к времени\r\n"
                     "'q' - выйти без сохранения\r\n"
                     "============================\r\n";
        HAL_UART_Transmit(&huart6, (uint8_t*)msg, strlen(msg), 100);
    }
```

```
void exit_edit_mode(bool save) {
    current_mode = MODE_MUSIC;
    set_green_led(false);
    set_yellow_led(is_track_playing);
    set_red_led(false);

    char msg[] = "\r\n=== ВОЗВРАТ В МУЗЫКАЛЬНЫЙ РЕЖИМ ===\r\n";
    HAL_UART_Transmit(&huart6, (uint8_t*)msg, strlen(msg), 100);
}

void process_edit_input(char input_char) {

    if (input_char == 'q') {
        char msg[32];
        sprintf(msg, "\r\nВыход. Нот: %lu\r\n", temp_size);
        HAL_UART_Transmit(&huart6, (uint8_t*)msg, strlen(msg), 100);

        current_mode = MODE_MUSIC;
        set_green_led(false);
        set_yellow_led(is_track_playing);
        set_red_led(false);
        return;
    }

    if (input_char == 'n') {
        if (is_editing_freq) {
            is_editing_freq = false;
            current_mode = MODE_EDIT_TIME;

            char msg[64];
            sprintf(msg, "\r\nВремя для %lu Гц: ", current_note_freq);
            HAL_UART_Transmit(&huart6, (uint8_t*)msg, strlen(msg), 100);
        }
        return;
    }

    if (input_char == 's') {
        if (current_mode == MODE_EDIT_TIME) {
            if (temp_size < 256) {
                temp_notes[temp_size] = current_note_freq;
                temp_timings[temp_size] = current_note_time;
                temp_size++;

                char msg[64];
                if (current_note_freq == 0) {
                    sprintf(msg, "\r\n[%lu] Пауза: %lu мс\r\n", temp_size,
current_note_time);
                } else {
                    sprintf(msg, "\r\n[%lu] Нота: %lu Гц, %lu мс\r\n",
```

```
                                temp_size, current_note_freq, current_note_time);
                }
                HAL_UART_Transmit(&huart6, (uint8_t*)msg, strlen(msg), 100);

                blink_leds(1);

                current_note_freq = 0;
                current_note_time = 0;
                is_editing_freq = true;
                current_mode = MODE_EDIT_FREQ;

                HAL_UART_Transmit(&huart6, (uint8_t*)"\r\nСлед. нота: ", 13,
100);
            } else {
                HAL_UART_Transmit(&huart6, (uint8_t*)"\r\nЛимит!\r\n", 9, 100);
            }
        }
        return;
    }

    if (input_char >= '0' && input_char <= '9') {
        uint8_t digit = input_char - '0';

        if (current_mode == MODE_EDIT_FREQ) {
            if (current_note_freq > 100000) {
                HAL_UART_Transmit(&huart6, (uint8_t*)"\r\nСлишком много!\r\n",
17, 100);
                return;
            }
            current_note_freq = current_note_freq * 10 + digit;

            char msg[32];
            sprintf(msg, "\r\nЧастота: %lu\r\n", current_note_freq);
            HAL_UART_Transmit(&huart6, (uint8_t*)msg, strlen(msg), 100);
        } else if (current_mode == MODE_EDIT_TIME) {
            if (current_note_time > 10000) {
                HAL_UART_Transmit(&huart6, (uint8_t*)"\r\nСлишком много!\r\n",
17, 100);
                return;
            }
            current_note_time = current_note_time * 10 + digit;

            char msg[32];
            sprintf(msg, "\r\nВремя: %lu\r\n", current_note_time);
            HAL_UART_Transmit(&huart6, (uint8_t*)msg, strlen(msg), 100);
        }
        return;
    }

    char msg[32];
    sprintf(msg, "\r\nНеизв: %c\r\n", input_char);
```

```
    HAL_UART_Transmit(&huart6, (uint8_t*)msg, strlen(msg), 100);
}

void process_settings_input(char input_char) {
    char response[128];

    if (input_char == 'q') {
        is_music_mode = true;
        is_settings_mode = false;
        set_green_led(false);
        set_yellow_led(false);
        set_red_led(false);

        char msg[] = "\r\n=== РЕЖИМ МУЗЫКИ ===\r\n";
        HAL_UART_Transmit(&huart6, (uint8_t*)msg, strlen(msg), 100);
        return;
    }

    if (input_char == 's') {
        if (temp_size > 0) {
            for (uint32_t i = 0; i < temp_size; i++) {
                custom_track_data[i] = temp_notes[i];
                custom_track_timing[i] = temp_timings[i];
            }
            custom_track_length = temp_size;

            char msg[64];
            sprintf(msg, "\r\nСохранено нот: %lu\r\n=== РЕЖИМ МУЗЫКИ ===\r\n",
temp_size);
            HAL_UART_Transmit(&huart6, (uint8_t*)msg, strlen(msg), 100);

            blink_leds(2);

            is_music_mode = true;
            is_settings_mode = false;
            set_green_led(false);
            set_yellow_led(false);
            set_red_led(false);
        } else {
            HAL_UART_Transmit(&huart6, (uint8_t*)"\r\nНет нот для
сохранения!\r\n", 26, 100);
        }
        return;
    }

    if (input_char == '?') {
        char help[] = "\r\n--- СПРАВКА ---\r\n"
                      "Формат: частота,длительность\r\n"
                      "Пример: 440,500 - нота Ля 440Гц на 500мс\r\n"
                      "Пример: 0,200 - пауза 200мс\r\n"
                      "'s' - сохранить, 'q' - выйти\r\n"
```

```
                        "----------------\r\n";
        HAL_UART_Transmit(&huart6, (uint8_t*)help, strlen(help), 100);
        return;
    }

    if (input_char == ',') {
        if (comma_detected) {
            HAL_UART_Transmit(&huart6, (uint8_t*)"\r\nОшибка: две
запятые!\r\n", 23, 100);
            note_value = 0;
            timing_value = 0;
            comma_detected = false;
        } else {
            comma_detected = true;
            HAL_UART_Transmit(&huart6, (uint8_t*)",", 1, 10);
        }
        return;
    }

    if (input_char == '\r') {
        if (temp_size < 256) {
            temp_notes[temp_size] = note_value;
            temp_timings[temp_size] = timing_value;
            temp_size++;

            char msg[64];
            if (note_value == 0) {
                sprintf(msg, "\r\n[%lu] Пауза: %lu мс\r\n", temp_size,
timing_value);
            } else {
                sprintf(msg, "\r\n[%lu] Нота: %lu Гц, %lu мс\r\n",
                        temp_size, note_value, timing_value);
            }
            HAL_UART_Transmit(&huart6, (uint8_t*)msg, strlen(msg), 100);

            set_yellow_led(true);
            HAL_Delay(50);
            set_yellow_led(false);

            note_value = 0;
            timing_value = 0;
            comma_detected = false;
        } else {
            HAL_UART_Transmit(&huart6, (uint8_t*)"\r\nЛимит нот (256)!\r\n",
19, 100);
        }
        return;
    }

    if (!isdigit((unsigned char)input_char)) {
        HAL_UART_Transmit(&huart6, (uint8_t*)"\r\nТолько цифры 0-9!\r\n", 20,
```

```
100);
            note_value = 0;
            timing_value = 0;
            comma_detected = false;
            return;
        }

        if (temp_size >= 256) {
            HAL_UART_Transmit(&huart6, (uint8_t*)"\r\nДостигнут лимит!\r\n", 18,
100);
            return;
        }

        if (comma_detected) {
            timing_value = timing_value * 10 + (input_char - '0');
        } else {
            note_value = note_value * 10 + (input_char - '0');
        }

        HAL_UART_Transmit(&huart6, (uint8_t*)&input_char, 1, 10);
}

void handle_key_press(int btn_index) {
    char received_char = key2char(btn_index);

    if (current_mode == MODE_TEST) {
        char msg[8];
        sprintf(msg, "[%c]\r\n", received_char);
        HAL_UART_Transmit(&huart6, (uint8_t*)msg, strlen(msg), 100);
        return;
    }

    if (current_mode == MODE_EDIT_FREQ || current_mode == MODE_EDIT_TIME) {
        process_edit_input(received_char);
        return;
    }

    char response[128];

    switch (received_char) {
        case '1': {
            sprintf(response, "\r\n1 - Led Zeppelin - Stairway to Heaven\r\n");
            HAL_UART_Transmit(&huart6, (uint8_t*)response, strlen(response),
100);
            stop_playback();
            start_track_playback(stairway_melody, stairway_delays,
                                 sizeof(stairway_melody) / sizeof(uint32_t));
            blink_leds(1);
            break;
        }
        case '2': {
```

```
            sprintf(response, "\r\n2 - Ssshhhiiiittt! - Серф Панк\r\n");
            HAL_UART_Transmit(&huart6, (uint8_t*)response, strlen(response),
100);
            stop_playback();
            start_track_playback(ssshhhiiiittt_melody, ssshhhiiiittt_delays,
                                 sizeof(ssshhhiiiittt_melody) / sizeof(uint32_t));
            blink_leds(2);
            break;
        }
        case '3': {
            sprintf(response, "\r\n3 - Boa - Duvet\r\n");
            HAL_UART_Transmit(&huart6, (uint8_t*)response, strlen(response),
100);
            stop_playback();
            start_track_playback(duvet_melody, duvet_delays,
                                 sizeof(duvet_melody) / sizeof(uint32_t));
            blink_leds(3);
            break;
        }
        case '4': {
            sprintf(response, "\r\n4 - Radiohead - No Surprises\r\n");
            HAL_UART_Transmit(&huart6, (uint8_t*)response, strlen(response),
100);
            stop_playback();
            start_track_playback(radiohead_melody, radiohead_delays,
                                 sizeof(radiohead_melody) / sizeof(uint32_t));
            blink_leds(4);
            break;
        }
        case '5': {
            if (custom_track_length > 0) {
                sprintf(response, "\r\n5 - Пользовательская мелодия (%lu
нот)\r\n", custom_track_length);
                HAL_UART_Transmit(&huart6, (uint8_t*)response,
strlen(response), 100);
                stop_playback();
                start_track_playback(custom_track_data, custom_track_timing,
custom_track_length);
                blink_leds(5);
            } else {
                sprintf(response, "\r\n!!! Нет пользовательской мелодии\r\n"
                                  "Нажмите 'e' для создания\r\n");
                HAL_UART_Transmit(&huart6, (uint8_t*)response,
strlen(response), 100);
            }
            break;
        }
        case 'x': {
            sprintf(response, "\r\n>>> Остановка\r\n");
            HAL_UART_Transmit(&huart6, (uint8_t*)response, strlen(response),
100);
```

```
            stop_playback();
            set_green_led(false);
            set_yellow_led(false);
            set_red_led(false);
            break;
        }
        case '?': {
            char menu[] = "\r\n=== МЕНЮ ===\r\n"
                          "1 - Stairway to Heaven\r\n"
                          "2 - Deftones\r\n"
                          "3 - Boa Duvet\r\n"
                          "4 - Radiohead\r\n"
                          "5 - Пользовательская\r\n"
                          "x - Стоп\r\n"
                          "e - Редактирование\r\n"
                          "? - Справка\r\n"
                          "=============\r\n";
            HAL_UART_Transmit(&huart6, (uint8_t*)menu, strlen(menu), 100);
            break;
        }
        case 'e': {
            if (current_mode != MODE_EDIT_FREQ && current_mode !=
MODE_EDIT_TIME) {
                enter_edit_mode();
            } else {
                char msg[64];
                if (is_editing_freq) {
                    sprintf(msg, "\r\nРежим: ввод частоты. Текущая: %lu
Гц\r\n", current_note_freq);
                } else {
                    sprintf(msg, "\r\nРежим: ввод времени. Текущее: %lu
мс\r\n", current_note_time);
                }
                HAL_UART_Transmit(&huart6, (uint8_t*)msg, strlen(msg), 100);
            }
            break;
        }
        default: {
            if (received_char != '!') {
                sprintf(response, "\r\n??? Неизвестная команда: %c\r\n",
received_char);
                HAL_UART_Transmit(&huart6, (uint8_t*)response,
strlen(response), 100);
            }
            break;
        }
    }
}

void save_custom_melody(void) {
    if (temp_size > 0) {
```

```
        for (uint32_t i = 0; i < temp_size; i++) {
            custom_track_data[i] = temp_notes[i];
            custom_track_timing[i] = temp_timings[i];
        }
        custom_track_length = temp_size;

        HAL_UART_Transmit(&huart6, (uint8_t*)"\r\n[SAVED]\r\n", 10, 10);

        set_green_led(true);
        HAL_Delay(50);
        set_green_led(false);
    } else {
        HAL_UART_Transmit(&huart6, (uint8_t*)"\r\n[NO NOTES]\r\n", 14, 10);
    }
}

void show_custom_melody(void) {
    if (temp_size > 0) {
        char msg[64];
        sprintf(msg, "\r\nМелодия: %lu нот\r\n", temp_size);
        HAL_UART_Transmit(&huart6, (uint8_t*)msg, strlen(msg), 100);

        HAL_Delay(10);

        uint32_t notes_to_show = temp_size;
        if (notes_to_show > 5) notes_to_show = 5;

        for (uint32_t i = 0; i < notes_to_show; i++) {
            if (temp_notes[i] == 0) {
                sprintf(msg, "[%lu] Пауза: %lu мс\r\n", i+1, temp_timings[i]);
            } else {
                sprintf(msg, "[%lu] Нота: %lu Гц, %lu мс\r\n", i+1,
temp_notes[i], temp_timings[i]);
            }
            HAL_UART_Transmit(&huart6, (uint8_t*)msg, strlen(msg), 100);
            HAL_Delay(5);
        }

        if (temp_size > 5) {
            HAL_UART_Transmit(&huart6, (uint8_t*)"... и еще нот\r\n", 14, 100);
        }
    } else {
        HAL_UART_Transmit(&huart6, (uint8_t*)"\r\nНет мелодии!\r\n", 15, 100);
    }

    HAL_Delay(10);
}


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);
        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;

    HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
    HAL_TIM_Base_Start_IT(&htim6);

    set_yellow_led(true);
}

void stop_playback(void) {
    is_track_playing = false;
    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_1, 0);
    track_position = 0;
    current_duration = 1;

    set_yellow_led(false);
}

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) {
                stop_playback();
                return;
            }

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

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

                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) {
                stop_playback();
            }
        }
    }
}
/* 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();
  MX_I2C1_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[] = "\r\n"
        "Музыкальная шкатулка с клавиатурой I2C\r\n"
        "Музыкальный режим:\r\n"
        "  1-4 - стандартные мелодии\r\n"
        "  5   - пользовательская мелодия\r\n"
        "  x   - остановка воспроизведения\r\n"
        "  e   - режим редактирования\r\n"
        "  ?   - справка\r\n"
        "\r\n"
        "Режим редактирования:\r\n"
        "  0-9 - ввод цифр\r\n"
        "  n   - следующий параметр\r\n"
        "  s   - сохранить ноту\r\n"
        "  q   - отмена/выход\r\n"
        "\r\n"
        "Кнопка PC15 - тестовый режим\r\n"
        "========================================\r\n";
      send_uart_line(welcome_message, strlen(welcome_message));
        start_track_playback(start_melody, start_delays,
                             sizeof(start_melody) / sizeof(uint32_t));
        blink_leds(3);
        uint8_t init_config = 0xFF;
          HAL_I2C_Mem_Write(&hi2c1, KB_I2C_WRITE_ADDRESS, KB_CONFIG_REG, 1,
&init_config, 1, 100);

          set_green_led(false);
          set_yellow_led(false);
          set_red_led(false);

          uint32_t last_mode_change = 0;
          bool mode_btn_pressed = false;

          buf_init(&rx_buffer);

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

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
      while (1)
        {
      static uint32_t last_cleanup = 0;
          if (HAL_GetTick() - last_cleanup > 1000) {
              __HAL_UART_CLEAR_FLAG(&huart6, UART_FLAG_TC);
              __HAL_UART_CLEAR_FLAG(&huart6, UART_FLAG_TXE);
              last_cleanup = HAL_GetTick();
```

```
            }
            bool current_btn_state = is_mode_btn_pressed();

            if (current_btn_state && !mode_btn_pressed) {
              mode_btn_pressed = true;
              last_mode_change = HAL_GetTick();
            }
            else if (!current_btn_state && mode_btn_pressed) {
              mode_btn_pressed = false;
              uint32_t press_duration = HAL_GetTick() - last_mode_change;

              if (press_duration > 50 && press_duration < 500) {
                if (current_mode == MODE_TEST) {
                  current_mode = MODE_MUSIC;
                  char msg[] = "\r\n=== ТЕСТОВЫЙ РЕЖИМ ВЫКЛЮЧЕН ===\r\n";
                  HAL_UART_Transmit(&huart6, (uint8_t*)msg, strlen(msg), 100);
                  set_red_led(false);
                } else {
                  previous_mode = current_mode;
                  current_mode = MODE_TEST;
                  char msg[] = "\r\n=== ТЕСТОВЫЙ РЕЖИМ ВКЛЮЧЕН ===\r\n"
                               "Нажимайте клавиши для теста\r\n"
                               "==============================\r\n";
                  HAL_UART_Transmit(&huart6, (uint8_t*)msg, strlen(msg), 100);
                  set_red_led(true);
                }
              }
              else if (press_duration >= 1000) {
                  save_custom_melody();
              }
            }

            int btn_index = get_pressed_btn_index();

            if (btn_index != -1) {
              handle_key_press(btn_index);
            }

            if (current_mode == MODE_MUSIC && is_track_playing) {
              static uint32_t last_blink = 0;
              if (HAL_GetTick() - last_blink > 200) {
                set_yellow_led(!HAL_GPIO_ReadPin(GPIOD, GPIO_PIN_14));
                last_blink = HAL_GetTick();
              }
            }
        /* USER CODE END WHILE */

        /* USER CODE BEGIN 3 */
      }
    }
```

```c
  /* 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)
    {
        set_green_led(true);
        set_yellow_led(true);
        set_red_led(true);
        HAL_Delay(200);
        set_green_led(false);
        set_yellow_led(false);
        set_red_led(false);
        HAL_Delay(200);
    }
  /* 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 функционал музыкальной шкатулки был перенесен на клавиатуру. Это позволило задействовать интерфейс UART исключительно для информационного вывода. Через интерфейс I2C реализована система сменных раскладок, обеспечивающая работу в различных режимах и возможность ввода пользовательских мелодий.