Университет ИТМО

Проектирование вычислительных систем

Лабораторная работа №3

Таймеры

ФИО студентов:
Готовко Алексей Владимирович
Руденко Илья Александрович
Вариант: 5
Направление подготовки: 09.03.04 (СППО)
Учебная группа: P34101
ФИО преподавателя: Пинкевич Василий Юрьевич

Санкт-Петербург
2024г.

Содержание

1 Цели работы

1. Получить базовые знания об устройстве и режимах работы таймеров в микроконтроллерах.
2. Получить навыки использования таймеров и прерываний от таймеров.
3. Получить навыки использования аппаратных каналов ввода-вывода таймеров.

2 Вариант задания

Реализовать музыкальную ритм-игру. С помощью звукоизлучателя воспроизводится последовательность, состоящая из звуков разной частоты («мелодия»). Каждый звук сопровождается зажиганием светодиода определенного цвета и с определенной яркостью (регулируется коэффициентом заполнения). Должно существовать взаимно однозначное соответствие между частотой звука и цветом/яркостью светодиода. Во время каждого звука/импульса светодиода игрок должен ввести символ, соответствующий текущей частоте звука или цвету/яркости. Чем больше звуков будет «угадано» правильно и на большей скорости игры, тем больше очков заработает игрок (система начисления очков – на усмотрение исполнителей).

Всего необходимо предусмотреть девять видов импульсов: зеленый, желтый и красный на 20%, 50% и 100% яркости. К ним следует подобрать звуки произвольных частот, легко отличимых одна от другой на слух. Предусмотреть одну стандартную последовательность импульсов длительностью не менее 20-ти элементов (простейший вариант – циклический перебор девяти импульсов). Когда последовательность заканчивается или досрочно останавливается игроком, в UART выводится количество набранных очков и «трассировка» нажатий, где отмечены правильные и неправильные нажатия. Отсутствие нажатия в течение импульса должно считаться неправильным нажатием.

Действия стенда при получении символов от компьютера:

Символ	**Действие**
«1» – «9»	Символы, которые надо вводить в течение игры во время соответствующих импульсов. Когда игра не запущена, при вводе данных символов соответствующие импульсы воспроизводятся на светодиодах и излучателе звука, а в UART выводится описание импульса, которому соответствует символ (цвет/яркость и частота звука).
«+»	Циклически переключать скорость игры (длительность каждого импульса): малая, средняя, быстрая. Необходимо подобрать длительности импульсов так, чтобы на малой скорости было комфортно набирать очки, а максимальная скорость не была «непроходимой».
«a»	Циклически переключать режим воспроизведения последовательности: светодиоды и звук, только светодиоды, только звук.
«Enter»	Запустить или досрочно остановить игру. При запуске должна выдерживаться пауза в три секунды после ввода символа, чтобы игрок успел подготовиться.

После ввода каждого символа в UART должно выводиться сообщение о том, какой импульс выбран или какой режим игры установлен.

Частота процессорного ядра – 60 МГц, частота ШИМ-сигнала для светодиодов – 500 Гц.

3 Описание программы

Опустим объяснение всей логики, реализованной и описанной в предыдущей лабораторной работе.

Для удобства расчета частоты ШИМ-сигнала на светодиоды установим Prescaler в 60, чтобы на выходе с него мы получали частоту в 1МГц. Если в ARR мы будем устанавливать значение, равное $\frac{1}{F_{target}}$, получим:

$$F_{PWM} = \frac{F_{clock}}{PSC \cdot ARR} = \frac{1}{ARR} = F_{target}$$

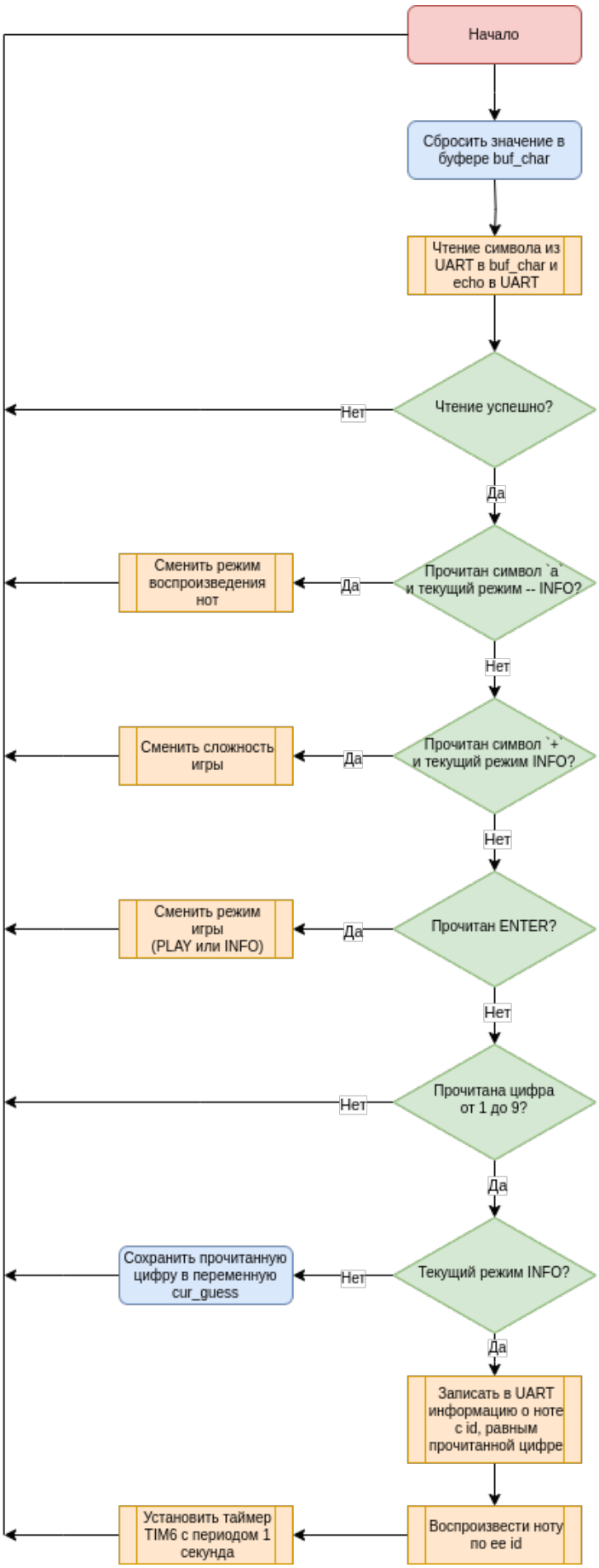
Для управления светодиодами использован таймер TIM4, для управления звукоизлучателем – TIM1, TIM6 – для отсчета необходимых промежутков времени.

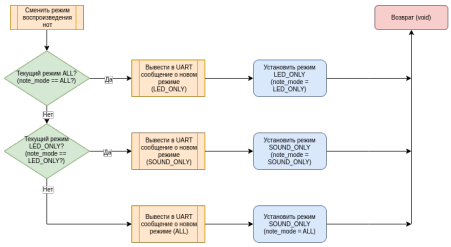
На каждой итерации программы будем выполнять следующую последовательность действий:

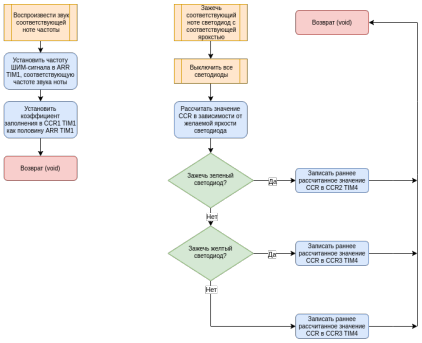
1. сбрасываем в 0 значение однобайтового буфера приема UART `buf_char`;
2. считываем символ из UART в `buf_char`;
3. если чтения не произошло, возвращаемся в начало цикла;
4. если прочитан символ 'a' и текущий режим INFO, то циклически меняем сложность игры и возвращаемся в начало;
5. если прочитан Enter, то циклически меняем режим игры и возвращаемся в начало;
6. если прочитана цифра от 1 до 9:
   (a) если режим игры GAME, то сохраняем текущий символ как ответ игрока в переменную `cur_guess`;
   (b) если режим игры INFO, то пишем в UART информацию о ноте с таким id, воспроизводим ее и устанавливаем таймер TIM6 в 1 секунду (столько времени будет проигрываться нота для ознакомления);

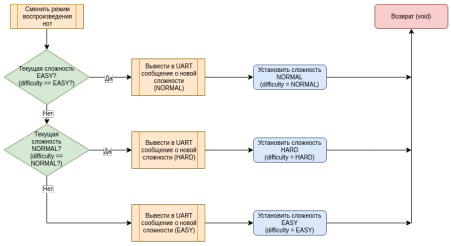
При начале игры будем засекать время для угадывания каждой ноты, соответствующее текущей сложности, с помощью таймера TIM6. По истечении времени будет вызвана callback-функция таймера, в которой будет происходить обработка в зависимости от текущего режима игры. По завершении игры таймер будем выключать.

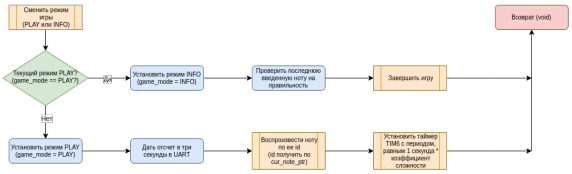
4 Блок-схема алгоритма

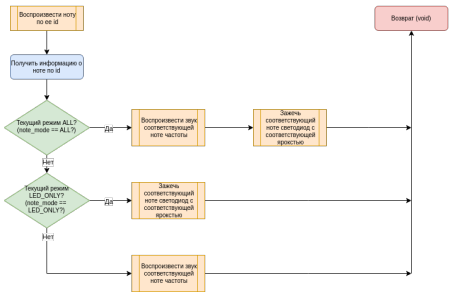


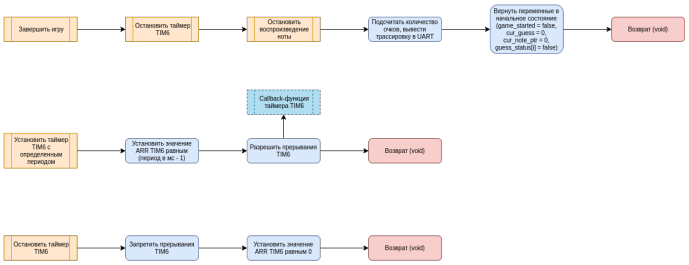

Сменить режим воспроизведения нот
Текущий режим ALL? (note_mode == ALL?)
Да
Вывести в UART сообщение о новом режиме (LED_ONLY)
Установить режим LED_ONLY (note_mode = LED_ONLY)
Нет
Текущий режим LED_ONLY? (note_mode == LED_ONLY?)
Да
Вывести в UART сообщение о новом режиме (SOUND_ONLY)
Установить режим SOUND_ONLY (note_mode = SOUND_ONLY)
Нет
Вывести в UART сообщение о новом режиме (ALL)
Установить режим SOUND_ONLY (note_mode = ALL)
Возврат (void)

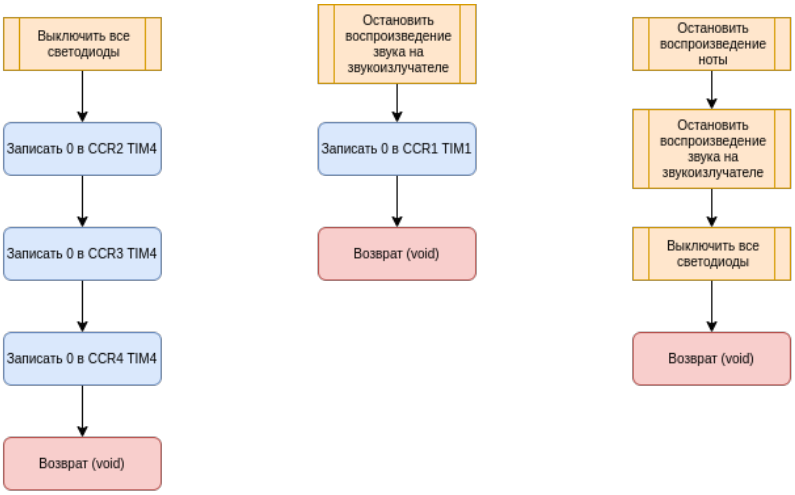

Воспроизвести звук соответствующей ноте частоты
Установить частоту ШИМ-сигнала в ARR TIM1, соответствующую частоте звука ноты
Установить коэффициент заполнения в CCR1 TIM1, как половину ARR TIM1
Возврат (void)
Зажечь соответствующий ноте светодиод с соответствующей яркостью
Выключить все светодиоды
Рассчитать значение CCR в зависимости от желаемой яркости светодиода
Зажечь зеленый светодиод?
Да
Записать раннее рассчитанное значение CCR в CCR2 TIM4
Нет
Зажечь желтый светодиод?
Да
Записать раннее рассчитанное значение CCR в CCR3 TIM4
Нет
Записать раннее рассчитанное значение CCR в CCR3 TIM4
Возврат (void)

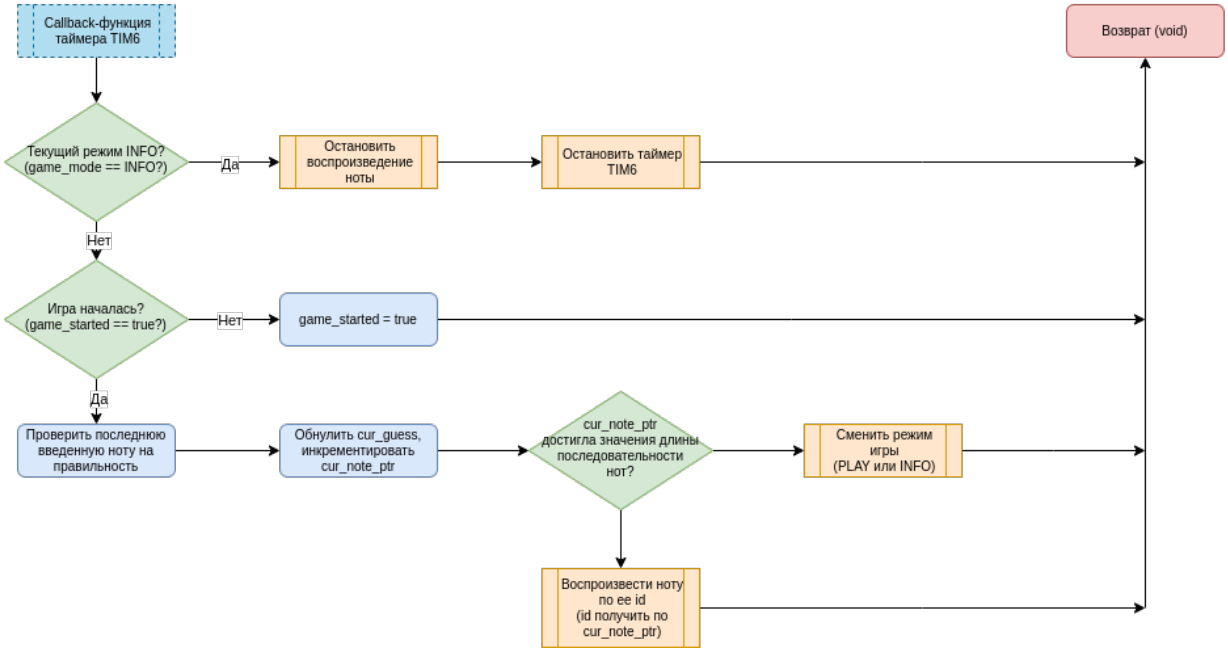

Сменить режим воспроизведения нот
Текущая сложность EASY? (difficulty == EASY?)
Да
Вывести в UART сообщение о новой сложности (NORMAL)
Установить сложность NORMAL (difficulty = NORMAL)
Нет
Текущая сложность NORMAL? (difficulty == NORMAL?)
Да
Вывести в UART сообщение о новой сложности (HARD)
Установить сложность HARD (difficulty = HARD)
Нет
Вывести в UART сообщение о новой сложности (EASY)
Установить сложность EASY (difficulty = EASY)
Возврат (void)


Сменить режим игры (PLAY или INFO)
Текущий режим PLAY? (game_mode == PLAY?)
Да
Установить режим INFO (game_mode = INFO)
Проверить последнюю введенную ноту на правильность
Завершить игру
Нет
Установить режим PLAY (game_mode = PLAY)
Дать отсчет в три секунды в UART
Воспроизвести ноту по ее id (id получить по cur_note_ptr)
Установить таймер TIM5 с периодом, равным 1 секунда * коэффициент сложности
Возврат (void)


Воспроизвести ноту по ее id
Получить информацию о ноте по id
Текущий режим ALL? (note_mode == ALL?)
Да
Воспроизвести звук соответствующей ноте частоты
Зажечь соответствующий ноте светодиод с соответствующей яркостью
Нет
Текущий режим LED_ONLY? (note_mode == LED_ONLY?)
Да
Зажечь соответствующий ноте светодиод с соответствующей яркостью
Нет
Воспроизвести звук соответствующей ноте частоты
Возврат (void)


Завершить игру
Остановить таймер TIM5
Остановить воспроизведение ноты
Подсчитать количество очков, вывести трассировку в UART
Вернуть переменные в начальное состояние (game_started = false, cur_guess = 0, cur_note_ptr = 0, guess_status[i] = false)
Возврат (void)
Callback-функция таймера TIM5
Установить таймер TIM5 с определенным периодом
Установить значение ARR TIM5 равным (период в мс - 1)
Разрешить прерывания TIM5
Возврат (void)
Остановить таймер TIM5
Запретить прерывания TIM5
Установить значение ARR TIM5 равным 0
Возврат (void)


Выключить все светодиоды
Записать 0 в CCR2 TIM4
Записать 0 в CCR3 TIM4
Записать 0 в CCR4 TIM4
Возврат (void)
Остановить воспроизведение звука на звукоизлучателе
Записать 0 в CCR1 TIM1
Возврат (void)
Остановить воспроизведение ноты
Остановить воспроизведение звука на звукоизлучателе
Выключить все светодиоды
Возврат (void)


Callback-функция таймера TIM6
Текущий режим INFO? (game_mode == INFO?)
Да
Остановить воспроизведение ноты
Остановить таймер TIM6
Возврат (void)
Нет
Игра началась? (game_started == true?)
Нет
game_started = true
Да
Проверить последнюю введенную ноту на правильность
Обнулить cur_guess, инкрементировать cur_note_ptr
cur_note_ptr достигла значения длины последовательности нот?
Сменить режим игры (PLAY или INFO)
Воспроизвести ноту по ее id (id получить по cur_note_ptr)

5 Код программы

Вспомогательные определения:

```
typedef enum {
        GREEN = 0,
        YELLOW,
        RED
} LED;

typedef enum {
        LOW = 20,
        MEDIUM = 50,
        HIGH = 100
} LEDBrightness;

typedef struct {
        LED led;
        LEDBrightness brightness;
        uint32_t frequency;
} Note;

typedef enum {
        INFO = 0,
        PLAY
} GameMode;

typedef enum {
        ALL = 0,
        LED_ONLY,
        SOUND_ONLY
} NoteMode;

typedef enum {
        EASY = 3,
        NORMAL = 2,
        HARD = 1
} Difficulty;


#define CLOCK_SCALED_FREQUENCY  1000000                 // frequency after scaling with PSC (supposed to be same on every timer in use)
#define LED_PWM_FREQUENCY       500
#define ENTER_ASCII             '\r'


uint16_t led_arr_value = CLOCK_SCALED_FREQUENCY / LED_PWM_FREQUENCY;

char newline[] = "\r\n";

char game_started_msg[] = "\n\rGame started!\n\r";
char game_finished_msg[] = "\n\rGame finished\n\r";

char note_mode_all_msg[] = "\n\rCurrent note mode: all\n\r";
char note_mode_led_msg[] = "\n\rCurrent note mode: led only\n\r";
```

```
char note_mode_sound_msg[] = "\n\rCurrent note mode: sound only\n\r";

char difficulty_easy_msg[] = "\n\rCurrent difficulty: easy\n\r";
char difficulty_normal_msg[] = "\n\rCurrent difficulty: normal\n\r";
char difficulty_hard_msg[] = "\n\rCurrent difficulty: hard\n\r";

char countdown_prepare_msg[] = "\n\rPrepare yourself!\n\r";
char countdown_3_msg[] = "\n\rStarting in 3...\n\r";
char countdown_2_msg[] = "\n\rStarting in 2...\n\r";
char countdown_1_msg[] = "\n\rStarting in 1...\n\r";

char guess_note_msg[] = "Guess the note: ";

bool game_started = false;

Difficulty difficulty = EASY;

char buf_char[2] = {'\0', '\0'};

GameMode game_mode = INFO;
NoteMode note_mode = ALL;

Note notes[] = {
                {GREEN,   LOW,    100},
                {GREEN,   MEDIUM, 200},
                {GREEN,   HIGH,   300},
                {YELLOW,  LOW,    400},
                {YELLOW,  MEDIUM, 500},
                {YELLOW,  HIGH,   600},
                {RED,     LOW,    700},
                {RED,     MEDIUM, 800},
                {RED,     HIGH,   900}
};

uint8_t note_sequence_length = 20;
uint8_t note_sequence[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 2};
bool guess_status[] = {false, false, false, false, false, false, false, false, false, false,
↪  false, false, false, false, false, false, false, false, false, false};
uint8_t cur_note_ptr = 0;
uint8_t cur_guess = 0;
```

Функции-драйверы:

```
void init_led_pwm(void) {
    htim4.Instance->ARR = led_arr_value;
}

void uart_write(char *data) {
    uint16_t size = strlen(data);
    HAL_UART_Transmit(&huart6, (uint8_t *) data, size, 100);
}

void uart_write_newline(void) {
    uart_write(newline);
}
```

```
13
14  void uart_read_char(void) {
15      HAL_StatusTypeDef status = HAL_UART_Receive(&huart6, (uint8_t *) buf_char, sizeof(char),
        ↪  100);
16      if (HAL_OK == status && process_char(buf_char))
17          uart_write(buf_char);
18  }
19
20  void play_sound(uint32_t *frequency) {
21      htim1.Instance->ARR = (1000000 / (*frequency)) - 1; // Set The PWM Frequency
22      htim1.Instance->CCR1 = (htim1.Instance->ARR >> 1); // Set Duty Cycle 50%
23  }
24
25  void mute(void) {
26      htim1.Instance->CCR1 = 0;
27  }
28
29  void disable_all_leds(void) {
30      htim4.Instance->CCR2 = 0;
31      htim4.Instance->CCR3 = 0;
32      htim4.Instance->CCR4 = 0;
33  }
34
35  void light_led(LED *led, LEDBrightness *brightness) {
36      disable_all_leds();
37      uint16_t ccr_value = CLOCK_SCALED_FREQUENCY / LED_PWM_FREQUENCY * (*brightness) / 100;
38      switch (*led) {
39          case GREEN:
40              htim4.Instance->CCR2 = ccr_value;
41              break;
42          case YELLOW:
43              htim4.Instance->CCR3 = ccr_value;
44              break;
45          case RED:
46              htim4.Instance->CCR4 = ccr_value;
47              break;
48      }
49  }
50
51  void play_note(Note *note) {
52      switch (note_mode) {
53          case ALL:
54              play_sound(&note->frequency);
55              light_led(&note->led, &note->brightness);
56              break;
57          case LED_ONLY:
58              light_led(&note->led, &note->brightness);
59              break;
60          case SOUND_ONLY:
61              play_sound(&note->frequency);
62              break;
63      }
64  }
65
66  void stop_note(void) {
```

```
    mute();
    disable_all_leds();
}

void uart_write_note_info(Note *note) {
    char buf[256];
    char *colour = note->led == GREEN ? "green" : (note->led == YELLOW ? "yellow" : "red");
    uint8_t brightness = note->brightness == LOW ? 20 : (note->brightness == MEDIUM ? 50 :
    ↪  100);

    sprintf(buf, "\n\rNote colour: %s\n\rNote brightness: %d\n\rNote frequency: %lu\n\r",
    ↪  colour, brightness, note->frequency);

    uart_write(buf);
}

void stop_timer(void) {
    HAL_TIM_Base_Stop_IT(&htim6);
    htim6.Instance->ARR = 0;
}

void set_timer_ms(uint32_t ms) {
    htim6.Instance->ARR = ms - 1;
    HAL_TIM_Base_Start_IT(&htim6);
}
```

Вспомогательные функции:

```
bool is_digit(char *c) {
    return '1' <= *c && *c <= '9';
}

bool process_char(char *c) {
    if (is_digit(&c[0]) || ENTER_ASCII == c[0])
        return true;
    if (('a' == c[0] || '+' == c[0]) && game_mode == INFO)
        return true;
    c[0] = '\0';
    return false;
}

void countdown_start_game(void) {
    uart_write(countdown_prepare_msg);
    uart_write(countdown_3_msg);
    HAL_Delay(1000);
    uart_write(countdown_2_msg);
    HAL_Delay(1000);
    uart_write(countdown_1_msg);
    HAL_Delay(1000);
}

void finish_game(void) {
    stop_timer();
    stop_note();

```

```
    uart_write_newline();
    uart_write(game_finished_msg);

    char buf[256];
    uint8_t points = 0;
    for (uint8_t i = 0; i < note_sequence_length; i++) {
        sprintf(buf, "Note %d: %s\n\r", i + 1, guess_status[i] ? "correct" : "wrong");
        uart_write(buf);
        points += (guess_status[i] * (4 - difficulty));

        guess_status[i] = false;
    }

    sprintf(buf, "Your score: %d\n\r", points);
    uart_write(buf);

    cur_note_ptr = 0;
    cur_guess = 0;
    game_started = false;
}

void switch_game_mode(void) {
    switch (game_mode) {
        case INFO:
            game_mode = PLAY;
            countdown_start_game();
            uart_write(game_started_msg);
            uart_write(guess_note_msg);
            play_note(&notes[note_sequence[cur_note_ptr] - 1]);
            set_timer_ms(difficulty * 1000);
            break;
        case PLAY:
            game_mode = INFO;
            if (cur_note_ptr < note_sequence_length && cur_guess == note_sequence[cur_note_ptr])
                guess_status[cur_note_ptr] = true;
            finish_game();
            break;
    }
}

void switch_note_mode(void) {
    stop_note();
    switch (note_mode) {
        case ALL:
            note_mode = LED_ONLY;
            uart_write(note_mode_led_msg);
            break;
        case LED_ONLY:
            note_mode = SOUND_ONLY;
            uart_write(note_mode_sound_msg);
            break;
        case SOUND_ONLY:
            note_mode = ALL;
            uart_write(note_mode_all_msg);
            break;
```

```
    }
}

void switch_difficulty(void) {
    switch (difficulty) {
        case EASY:
            difficulty= NORMAL;
            uart_write(difficulty_normal_msg);
            break;
        case NORMAL:
            difficulty= HARD;
            uart_write(difficulty_hard_msg);
            break;
        case HARD:
            difficulty= EASY;
            uart_write(difficulty_easy_msg);
            break;
    }
}
```

Основной цикл программы:

```
HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_2);
HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_3);
HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_4);

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

init_led_pwm();
stop_timer();

while (1)
{
    buf_char[0] = '\0';
    uart_read_char();
    if (buf_char[0] == '\0')
        continue;

    if (buf_char[0] == ENTER_ASCII) {
        switch_game_mode();
        continue;
    }

    if (is_digit(&buf_char[0])) {
        if (game_mode == INFO) {
            uint8_t id = buf_char[0] - '0' - 1;
            play_note(&notes[id]);
            uart_write_note_info(&notes[id]);
            set_timer_ms(1000);
        } else {
            cur_guess = buf_char[0] - '0';
        }
        continue;
    }
```

```
34
35      if (buf_char[0] == 'a' && game_mode == INFO) {
36          switch_note_mode();
37          continue;
38      }
39
40      if (buf_char[0] == '+' && game_mode == INFO) {
41          switch_difficulty();
42          continue;
43      }
44  }
```

6 Вывод

В результате выполнения лабораторной работы были получены базовые знания об устройстве и режимах работы таймеров в микроконтроллерах, навыки использования таймеров и прерываний от таймеров и навыки использования аппаратных каналов ввода-вывода таймеров.