Университет ИТМО
ПИиКТ

Лабораторная работа №4:
«Интерфейс I^2C и матричная клавиатура»

По дисциплине: Проектирование вычислительных систем

Вариант: 3

Выполнили:
Неграш А. В., P34301
Перевозчиков И. С., P34301

Преподаватель:
Пинкевич В. Ю.

Санкт-Петербург
2023

1. Задание

Разработать программу, которая использует интерфейс I2C для считывания нажатий 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. В режиме тестирования клавиатуры программа выводит в UART коды нажатых кнопок. В прикладном режиме программа обрабатывает нажатия кнопок и выполняет действия в соответствии с вариантом задания.

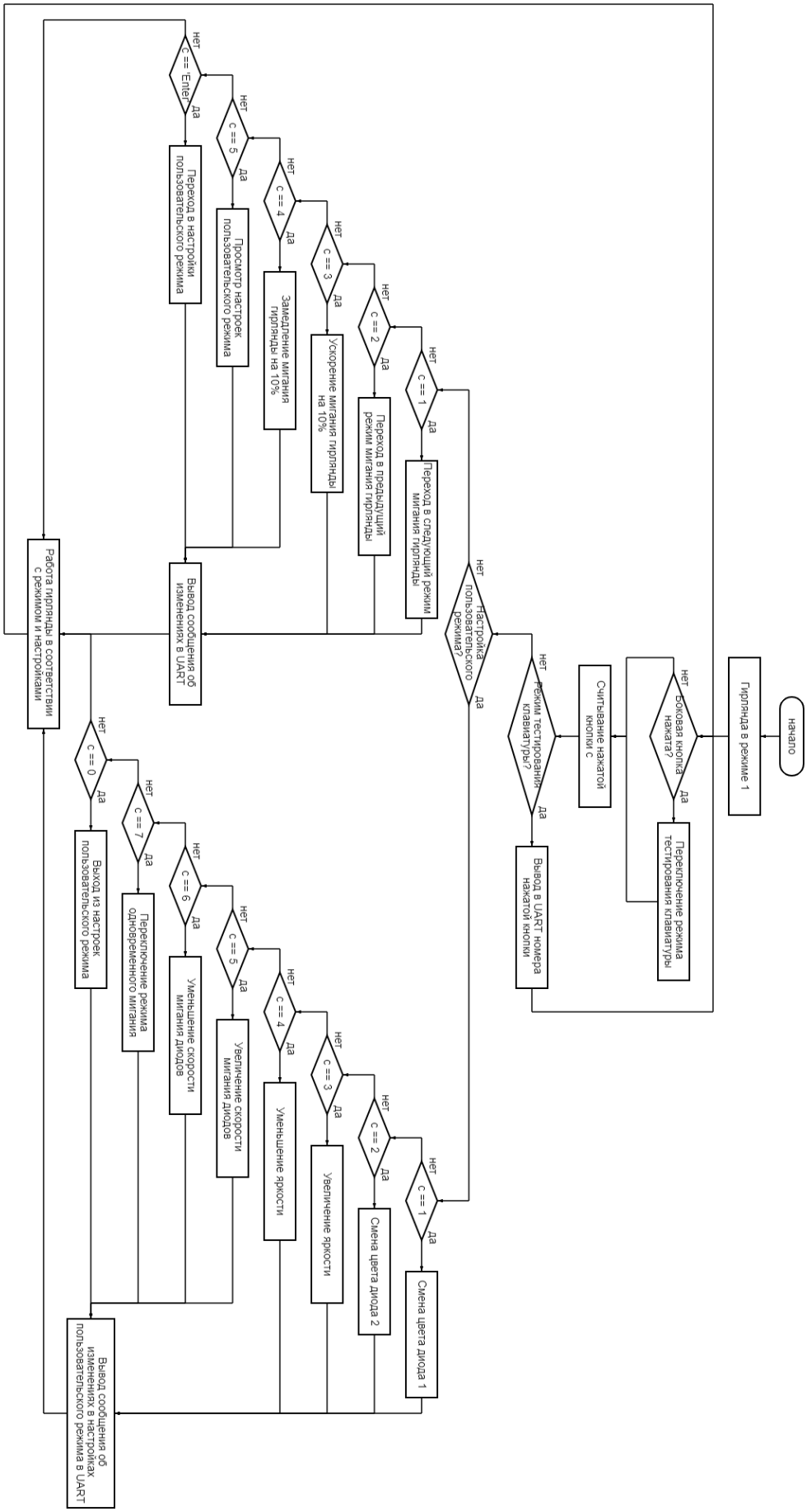
Вариант 3:

Реализовать имитатор гирлянды с плавными включениями/выключениями светодиодов и переходами между цветами. Должно быть предусмотрено четыре разных предустановленных режима анимации и один пользовательский режим, который можно настраивать. В каждом режиме должно быть задействовано минимум два светодиода, при этом не обязательно, чтобы они могли гореть одновременно. Действия стенда при получении символов от компьютера:

Символ	**Действие**
«1»	Переход на следующий режим
«2»	Возврат к предыдущему режиму
«3»	Ускорение воспроизведения анимации на 10% от текущей скорости
«4»	Замедление воспроизведения анимации на 10% от текущей скорости
«5»	Вывод в UART параметров текущего сохраненного пользовательского режима.
«Enter»	Вход в меню настройки.
На усмотрение исполнителей	Ввод параметров пользовательского режима: цвет светодиода, яркость, необходимость плавного перехода, длительность, конец последовательности и т.п.

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

2. Блок-схема прикладного алгоритма



3. Исходный код

```
#define BUF_SIZE 1024
#define COLOR_GREEN 1
#define COLOR_RED 2
#define COLOR_YELLOW 3

#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)

static char __my_buf[128];

#define my_printf_macro( ... ) \
	do { \
		sprintf( __my_buf, __VA_ARGS__ ); \
		HAL_UART_Transmit( &huart6, (uint8_t *) __my_buf, strlen(__my_buf),
100); \
	} while ( 0 )

struct queue {
	char buff[BUF_SIZE];
	int first;
	int last;
	int size;
	bool empty;

} terminal_queue, stand_queue;

int top(struct queue *q, char *s) {
	if (q->size == 0) {
		q->empty = true;
		return 0;
	}
	*s = q->buff[q->first];
	return 1;
}

void push_many(struct queue *q, char *el) {
	uint64_t size = strlen(el);

	if (q->first + size + 1 > BUF_SIZE) {
		q->first = 0;
	}

	strcpy(&q->buff[q->first], el);
    q->first += size + 1;

    if (q->first == BUF_SIZE) {
        q->first = 0;
    }

    q->empty = false;
}

int push_one(struct queue *q, char s) {
	q->buff[q->last] = s;
```

```
      if (q->size < BUF_SIZE) {
        q->size += 1;
        q->last = (q->last + 1) % BUF_SIZE;
        q->empty = false;

        return 1;
      }
      return 0;
}

bool pop_many(struct queue *q, char * el) {
    if (q->empty) {
        return false;
    }

    uint64_t size = strlen(&q->buff[q->last]);

    strcpy(el, &q->buff[q->last]);
    q->last += size + 1;

    if (q->last == BUF_SIZE || q->last == '\0') {
        q->last = 0;
    }

    if (q->last == q->first) {
        q->empty = true;
    }
    return true;
}

int pop_one(struct queue *q, char *s) {
      if (q->empty) {
            q->size = 0;
            return 0;
      }

      *s = q->buff[q->first];
      if (q->size != 0) {
            q->size -= 1;
      }
      q->first = (q->first + 1) % BUF_SIZE;
      if (q->size == 0) q->empty = true;
      if (q->first==q->last) q->empty = true;

      return 1;
}

struct led_state{
  int color;
      int turned_on;
      char c;
      int switch_state_time;
} led_state_green, led_state_red, led_state_yellow;

void green_led_off() {
      led_state_green.turned_on = 0;
      htim4.Instance->CCR2 = 0;
}

void green_led_on(){
      led_state_green.turned_on = 1;
```

```
	led_state_green.switch_state_time = HAL_GetTick();
	htim4.Instance->CCR2 = 1000;
}

void red_led_off() {
	led_state_red.turned_on = 0;
	htim4.Instance->CCR3 = 0;
	htim4.Instance->CCR4 = 0;
}

void red_led_on(){
	led_state_red.turned_on = 1;
	led_state_red.switch_state_time = HAL_GetTick();
	htim4.Instance->CCR4 = 1000;
	htim4.Instance->CCR3 = 0;
}

void yellow_led_off() {
	led_state_yellow.turned_on = 0;
	htim4.Instance->CCR3 = 0;
	htim4.Instance->CCR4 = 0;
}

void yellow_led_on(){
	led_state_yellow.turned_on = 1;
	led_state_yellow.switch_state_time = HAL_GetTick();
	htim4.Instance->CCR4 = 0;
	htim4.Instance->CCR3 = 1000;
}

char readable[2] = {"\0\0"};

void read_symbol_with_ints(){
	HAL_UART_Receive_IT(&huart6, readable, sizeof(char));
}

int settings_mode = 0;
// 1-4, user = 5
int mode = 1;

int user_mode_color_1 = COLOR_GREEN;
int user_mode_color_2 = COLOR_RED;
int bright = 1000;
int bright_user = 1000;
int bright_smooth = 1000;
int duration = 1000; // 1 sec
int duration_user = 1000;
bool use_smooth = true;
bool two_same = false;
bool transmit_busy = false;
uint32_t last_pressing_time = 0;
int last_pressed_btn_index = -1;
bool is_test_keyboard_mode = false;

void init_led_state(){
  led_state_green.color = COLOR_GREEN;
  led_state_green.switch_state_time = HAL_GetTick();
  green_led_off();

  led_state_red.color = COLOR_RED;
  led_state_red.switch_state_time = HAL_GetTick();
```

```
  red_led_off();

  led_state_yellow.color = COLOR_YELLOW;
  led_state_yellow.switch_state_time = HAL_GetTick();
  yellow_led_off();

  bright_smooth = bright_user;
}

void transmit_uart_one(char c) {
      if (transmit_busy) {
            push_many(&stand_queue, &c);
      }
      else {
         HAL_UART_Transmit_IT(&huart6, &c, 1);
         transmit_busy = true;
      }
}

void transmit_uart(char *buf, size_t size) {
      if (transmit_busy) {
            push_many(&stand_queue, buf);
      }
      else {
         HAL_UART_Transmit_IT(&huart6, buf, size);
         transmit_busy = true;
      }
}

void transmit_current_mode(){
  switch (mode) {
    case 1:
              transmit_uart(&change_mode_1, sizeof(change_mode_1));
      break;
    case 2:
              transmit_uart(&change_mode_2, sizeof(change_mode_2));
      break;
    case 3:
              transmit_uart(&change_mode_3, sizeof(change_mode_3));
      break;
    case 4:
              transmit_uart(&change_mode_4, sizeof(change_mode_4));
      break;
    case 5:
              transmit_uart(&change_mode_5, sizeof(change_mode_5));
      break;
  }
}

void send_user_settings_str(){
      my_printf_macro("\r\nFirst LED color: %d\n\rSecond LED color: %d\n\rAnimation
speed: %d\r\nBoth mode: %d\r\nBrightness: %d\r\n\r\n\0",
                  user_mode_color_1, user_mode_color_2, duration_user, two_same,
bright_user);
}


void change_settings(){
      char c;
      int popped = pop_one(&terminal_queue, &c);
      if (!is_test_keyboard_mode) {
```

```
                    if (popped) {
                      if (settings_mode) {
                            switch (c) {
                                  case '0':
                                        settings_mode = 0;
                                        transmit_uart(&exit_user_mode_msg,
sizeof(exit_user_mode_msg));
                                        break;
                                  case '1':
                                        //Сменить цвет светодиода 1.
                                        if(user_mode_color_1 == 3) user_mode_color_1
= 1;
                                        else user_mode_color_1++;
                                        transmit_uart(&led_1_change_msg,
sizeof(led_1_change_msg));
                                        break;
                                  case '2':
                                        //Сменить цвет светодиода 2
                                        if(user_mode_color_2 == 3) user_mode_color_2
= 1;
                                        else user_mode_color_2++;
                                        transmit_uart(&led_2_change_msg,
sizeof(led_2_change_msg));
                                        break;
                                  case '3':
                                        //Увеличить яркость
                                        if(bright_user == 1000) bright_user = 100;
                                        else bright_user += 100;
                                        transmit_uart(&bright_plus_msg,
sizeof(bright_plus_msg));
                                        break;
                                  case '4':
                                        //Уменьшить яркость
                                        if(bright_user == 100) bright_user = 1000;
                                        else bright_user -= 100;
                                        transmit_uart(&bright_minus_msg,
sizeof(bright_minus_msg));
                                        bright_smooth = bright_user;
                                        break;
                                  case '5':
                                        //Увеличить длительность
                                        duration_user += 100;
                                        transmit_uart(&duration_plus_msg,
sizeof(duration_plus_msg));
                                        break;
                                  case '6':
                                        //Уменьшить длительность
                                        duration_user -= 100;
                                        transmit_uart(&duration_minus_msg,
sizeof(duration_minus_msg));
                                        break;
                                  case '7':
                                        two_same = !two_same;
                                        transmit_uart(&smooth_change_msg,
sizeof(smooth_change_msg));
                                        break;
                                  default:
                                        transmit_uart(&invalid_option_msg,
sizeof(invalid_option_msg));
                                        break;
                            }
```

```
                        } else {
                                switch (c) {
                                my_printf_macro("c is %d", c);
                                case '1':
                                        if (mode < 5) mode++; else mode = 1;
                                        duration = 1000;
                                        transmit_current_mode();
                                        break;
                                case '2':
                                        if (mode != 1) mode--; else mode = 5;
                                        duration = 1000;
                                        transmit_current_mode();
                                        break;
                                case '3':
                                        if (duration < 2000) duration = duration -
duration*0.1; else duration = 1000;
                                        transmit_uart(&speed_plus_msg,
sizeof(speed_plus_msg));
                                        break;
                                case '4':
                                        if (duration > 100) duration = duration +
duration*0.1; else duration = 1900;
                                        transmit_uart(&speed_minus_msg,
sizeof(speed_minus_msg));
                                        break;
                                case '5':
                                        send_user_settings_str();
                                        break;
                                case '\n':
                                case '\r':
                                        settings_mode = 1;
                                        transmit_uart(&options_menu_msg,
sizeof(options_menu_msg));
                                        break;
                                default:
                                        transmit_uart(&invalid_option_msg,
sizeof(invalid_option_msg));
                                        break;
                                }

                        }
                    }
        }

}

void turn_on(struct led_state* ls){
  if (ls->color == COLOR_GREEN) green_led_on();
  else if (ls->color == COLOR_RED) red_led_on();
  else if (ls->color == COLOR_YELLOW) yellow_led_on();
}

void turn_off(struct led_state* ls){
  if (ls->color == COLOR_GREEN) green_led_off();
  else if (ls->color == COLOR_RED) red_led_off();
  else if (ls->color == COLOR_YELLOW) yellow_led_off();
}

void set_brightness(struct led_state* ls, int bright){
  if (ls->color == COLOR_GREEN) {
    htim4.Instance->CCR2 = bright;
```

```
    htim4.Instance->CCR3 = 0;
    htim4.Instance->CCR4 = 0;
  }
  else if (ls->color == COLOR_RED) {
    htim4.Instance->CCR4 = bright;
    htim4.Instance->CCR2 = 0;
    htim4.Instance->CCR3 = 0;
  }
  else if (ls->color == COLOR_YELLOW) {
    htim4.Instance->CCR3 = bright;
    htim4.Instance->CCR2 = 0;
    htim4.Instance->CCR4 = 0;
  }
}

void led(){
      int now = HAL_GetTick();
      struct led_state *first;
      struct led_state *second;
      int dur = duration;
      int brightness = bright;

      if ( mode == 1 || mode == 4) {
      // зеленый красный
      first = &led_state_green;
      second = &led_state_red;
      } else if (mode == 2) {
      // зеленый желтый
      first = &led_state_green;
      second = &led_state_yellow;
      } else if (mode == 3) {
      // красный желтый
      first = &led_state_red;
      second = &led_state_yellow;
      } else if (mode == 5) {
            if (user_mode_color_1 == COLOR_GREEN) {
             first = &led_state_green;
            } else if (user_mode_color_1 == COLOR_RED) {
             first = &led_state_red;
            } else if (user_mode_color_1 == COLOR_YELLOW) {
             first = &led_state_yellow;
            }

            if (user_mode_color_2 == COLOR_GREEN) {
             second = &led_state_green;
            } else if (user_mode_color_2 == COLOR_RED) {
             second = &led_state_red;
            } else if (user_mode_color_2 == COLOR_YELLOW) {
             second = &led_state_yellow;
            }
            dur = duration_user;
            brightness = bright_user;
      }

      int first_on = first->turned_on;
      int second_on = second->turned_on;
      int first_time = first->switch_state_time;
      int second_time = second->switch_state_time;

      // если первый горит и уже пора выключать, выключаем
      if (first_on && (first_time + dur) <= now) {
```

```
      turn_off(first);
      }

      // если второй горит и уже пора выключать, выключаем
      if (second_on && (second_time + dur) <= now) {
      turn_off(second);
      }

    // если никто не горит, включаем тот, который горел позже
      if (!first_on && !second_on) {
            // надо включить первый
       if (first_time <= second_time && (second_time + dur) <= now) {
            turn_on(first);
            if ( mode == 5 ) set_brightness(first, bright_user);
            // если в режиме с двумя огоньками одновременно, тогда включаем оба
            if (mode == 4 || two_same ) turn_on(second);
            // надо включить второй
       } else if (second_time <= first_time && (first_time + dur) <= now) {
            turn_on(second);
            if ( mode == 5  ) set_brightness(second, bright_user);
            // если в режиме с двумя огоньками одновременно, тогда включаем оба
            if (mode == 4 || two_same ) turn_on(first);
       }
      }
}

// коллбэк на прием
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart) {
 if (huart->Instance == USART6) {
      char c = readable[0];
      push_one(&terminal_queue, c);
      change_settings();
 }
}

// коллбэк на передачу
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart) {
 if (huart->Instance == USART6) {
      char buf[1024];
      if (pop_many(&stand_queue, buf)){
               transmit_busy = false;
               HAL_UART_Transmit_IT( &huart6, &buf, strlen(buf) );
      } else {
               transmit_busy = false;
      }
 }
}

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

int get_pressed_btn_index(){
      const uint32_t t = HAL_GetTick();
      if (t - last_pressing_time < KB_KEY_DEBOUNCE_TIME) return -1;
      int index = -1;
      uint8_t reg_buffer = ~0;
      uint8_t tmp = 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 buf = ~((uint8_t) (1 << row));
            HAL_I2C_Mem_Write(&hi2c1, KB_I2C_WRITE_ADDRESS, KB_CONFIG_REG, 1, &buf,
1, KB_KEY_DEBOUNCE_TIME);
            HAL_Delay(100);
            HAL_I2C_Mem_Read(&hi2c1, KB_I2C_READ_ADDRESS, KB_INPUT_REG, 1,
&reg_buffer, 1, KB_KEY_DEBOUNCE_TIME);
            switch(reg_buffer >> 4){
                  case 6: index = row * 3 + 1; break;
                  case 5: index = row * 3 + 2; break;
                  case 3: index = row * 3 + 3; break;
            }
      }
      if (index != -1) last_pressing_time = t;
      if (index == last_pressed_btn_index){
            return -1;
      }
      last_pressed_btn_index = index;
      return index;
}

bool is_btn_press() {
       return HAL_GPIO_ReadPin(GPIOC,GPIO_PIN_15) == 0;
 }

bool last_btn_state = false;

char kb_btn_1[] = {"\r\n1\r\n"};
char kb_btn_2[] = {"\r\n2\r\n"};
char kb_btn_3[] = {"\r\n3\r\n"};
char kb_btn_4[] = {"\r\n4\r\n"};
char kb_btn_5[] = {"\r\n5\r\n"};
char kb_btn_6[] = {"\r\n6\r\n"};
char kb_btn_7[] = {"\r\n7\r\n"};
char kb_btn_8[] = {"\r\n8\r\n"};
char kb_btn_9[] = {"\r\n9\r\n"};
char kb_btn_10[] = {"\r\n10\r\n"};
char kb_btn_11[] = {"\r\n11\r\n"};
char kb_btn_12[] = {"\r\n12\r\n"};

/* 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_TIM4_Init();
  MX_TIM6_Init();
  MX_I2C1_Init();
  /* USER CODE BEGIN 2 */
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
    HAL_NVIC_EnableIRQ(USART6_IRQn);
      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(&htim6);

      init_led_state();

      while (1)
    {
            // читаем символ с клавиатуры
            read_symbol_with_ints();

            bool btn_state = is_btn_press();
            if (last_btn_state && !btn_state){
                  is_test_keyboard_mode = !is_test_keyboard_mode;
                  if (is_test_keyboard_mode) {
                        transmit_uart(test_keyboard_on, sizeof(test_keyboard_on));
                  } else {
                        transmit_uart(test_keyboard_off,
sizeof(test_keyboard_off));
                  }
            }
            last_btn_state = btn_state;
            int btn_index = get_pressed_btn_index();
            if (btn_index != -1) {
                  char received_char = -1;
                  switch(btn_index){
                        case 1:
                              if (is_test_keyboard_mode) transmit_uart(&kb_btn_1,
sizeof(kb_btn_1));
                              received_char = '1';
                          break;
                      case 2:
                          if (is_test_keyboard_mode) transmit_uart(&kb_btn_2,
sizeof(kb_btn_2));
                          received_char = '2';
                          break;
                      case 3:
                          if (is_test_keyboard_mode) transmit_uart(&kb_btn_3,
sizeof(kb_btn_3));
```

```
                        received_char = '3';
                      break;
                    case 4:
                      if (is_test_keyboard_mode) transmit_uart(&kb_btn_4,
sizeof(kb_btn_4));
                        received_char = '4';
                      break;
                    case 5:
                      if (is_test_keyboard_mode) transmit_uart(&kb_btn_5,
sizeof(kb_btn_5));
                        received_char = '5';
                      break;
                    case 6:
                      if (is_test_keyboard_mode) transmit_uart(&kb_btn_6,
sizeof(kb_btn_6));
                        else if (settings_mode) received_char = '6';
                        else received_char = '\n';
                      break;
                    case 7:
                      if (is_test_keyboard_mode) transmit_uart(&kb_btn_7,
sizeof(kb_btn_7));
                        received_char = '7';
                      break;
                    case 8:
                      if (is_test_keyboard_mode) transmit_uart(&kb_btn_8,
sizeof(kb_btn_8));
                        received_char = '8';
                      break;
                    case 9:
                      if (is_test_keyboard_mode) transmit_uart(&kb_btn_9,
sizeof(kb_btn_9));
                        received_char = '9';
                      break;
                    case 10:
                      if (is_test_keyboard_mode) transmit_uart(&kb_btn_10,
sizeof(kb_btn_10));
                      break;
                    case 11:
                      if (is_test_keyboard_mode) transmit_uart(&kb_btn_11,
sizeof(kb_btn_11));
                        received_char = '0';
                      break;
                    case 12:
                      if (is_test_keyboard_mode) transmit_uart(&kb_btn_12,
sizeof(kb_btn_12));
                        break;
                }

                if (!is_test_keyboard_mode && received_char != -1) {
                      push_one(&terminal_queue, received_char);
                      change_settings();
                }
            }

    /* USER CODE END WHILE */

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

4. Вывод

Итак, в процессе выполнения данной лабораторной работы мы продолжили ознакомление с устройством стенда SDK 1.1M и изучили работу с матричной клавиатурой и интерфейсом I^2C, а также модифицировали реализованную в лабораторной работе №3 программу, имитирующую гирлянду с различными режимами, в которой продемонстрировали полученные знания.