УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерной техники

Проектирование вычислительных систем

Лабораторная работа №3: Таймеры и интерфейс I2C
Вариант 2

Преподаватель: Пинкевич Василий Юрьевич

Выполнили: Тарасов Александр Станиславович, Кульбако Артемий Юрьевич
P34112

Санкт-Петербург
2022

Задание

Реализовать настраиваемый пульт включения разных режимов горения светодиодов. По нажатию кнопок клавиатуры выполняются следующие действия:

Код кнопки	Действие
1-9	Зажигание светодиода в соответствии с режимом. Предыдущий режим горения отключается, новый режим держится до переключения на следующий режим. Режимы по умолчанию: 1 – зеленый, 10% яркости 2 – зеленый, 40% яркости 3 – зеленый, 100% яркости 4 – желтый, 10% яркости 5 – желтый, 40% яркости 6 – желтый, 100% яркости 7 – красный, 10% яркости 8 – красный, 40% яркости 9 – красный, 100% яркости
10	Отключить текущий режим (погасить все светодиоды).
11	Войти в меню настройки.
12	Выйти из меню настройки.

По нажатию каждой кнопки в UART должно выводиться сообщение о том, какой режим активирован, или текущие настройки, вводимые в меню.

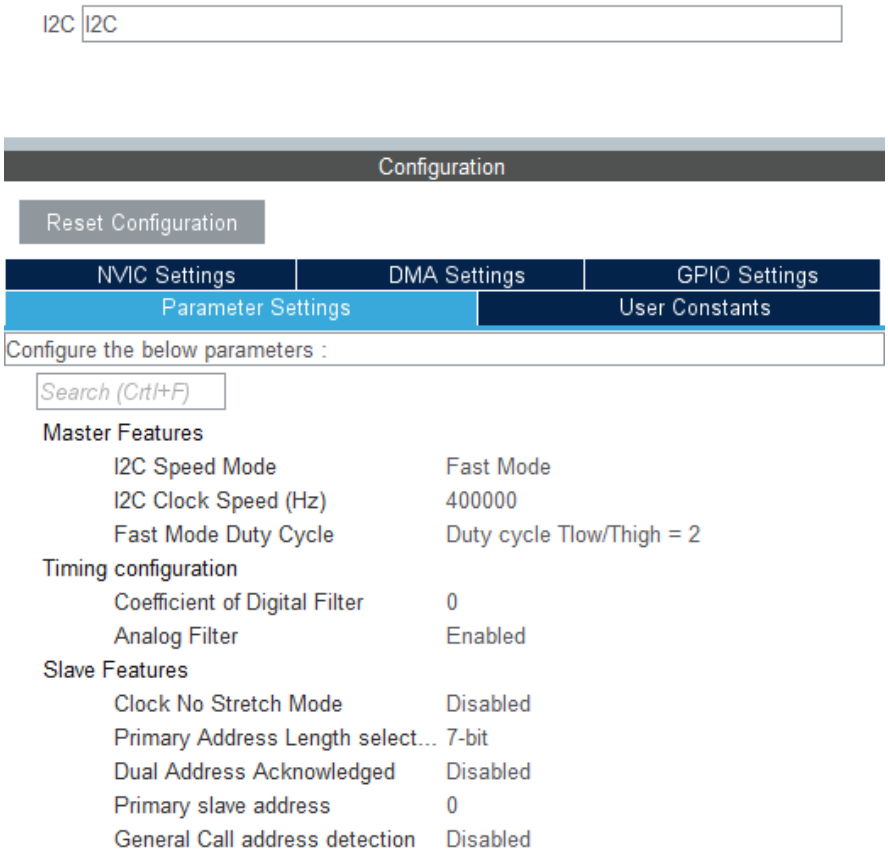
После входе в меню настройки сначала надо нажать кнопку, привязанный к которой режим требуется изменить, далее кнопками 1 – 3 выбирается светодиод (зеленый, желтый, красный) и кнопками 4, 5 – коэффициент заполнения от 0 до 100% с шагом 10%. По нажатию кнопки выхода из меню новый режим сохраняется.

Настройка

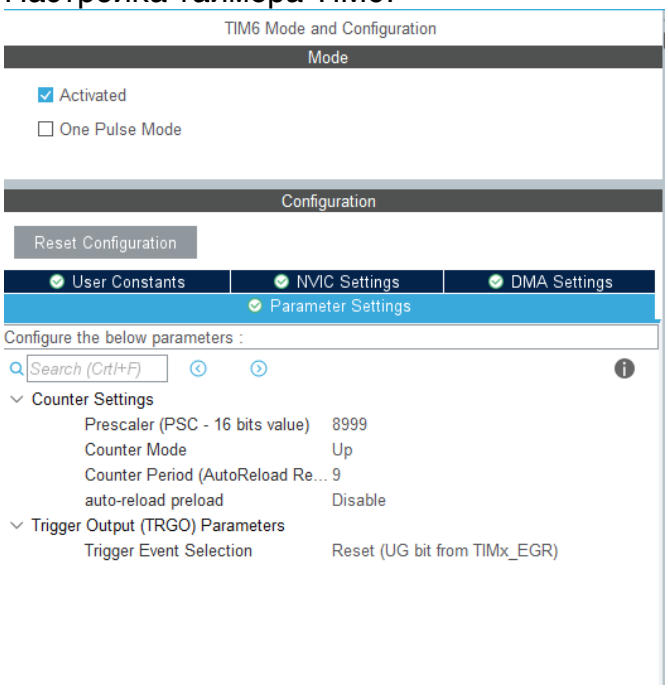
Микросхема PCA9538 является внешним расширителем GPIO-портов ввода/вывода, который подключается к микроконтроллеру по интерфейсу I2C. Данная микросхема реализует подключение до 8 сигналов, каждый из которых может быть настроен как вход или выход.

Адресный байт содержит постоянную часть и два бита, следовательно, возможно одновременное подключение на шину I2C до 4 одинаковых микросхем. В SDK-1.1M имеется два устройства с адресами 0xE0 (входы прерываний устройств и др.) и 0xE2 (клавиатура). Последний бит адресного байта подчиненного устройства определяет операцию (чтение или запись), которая должна быть выполнена.

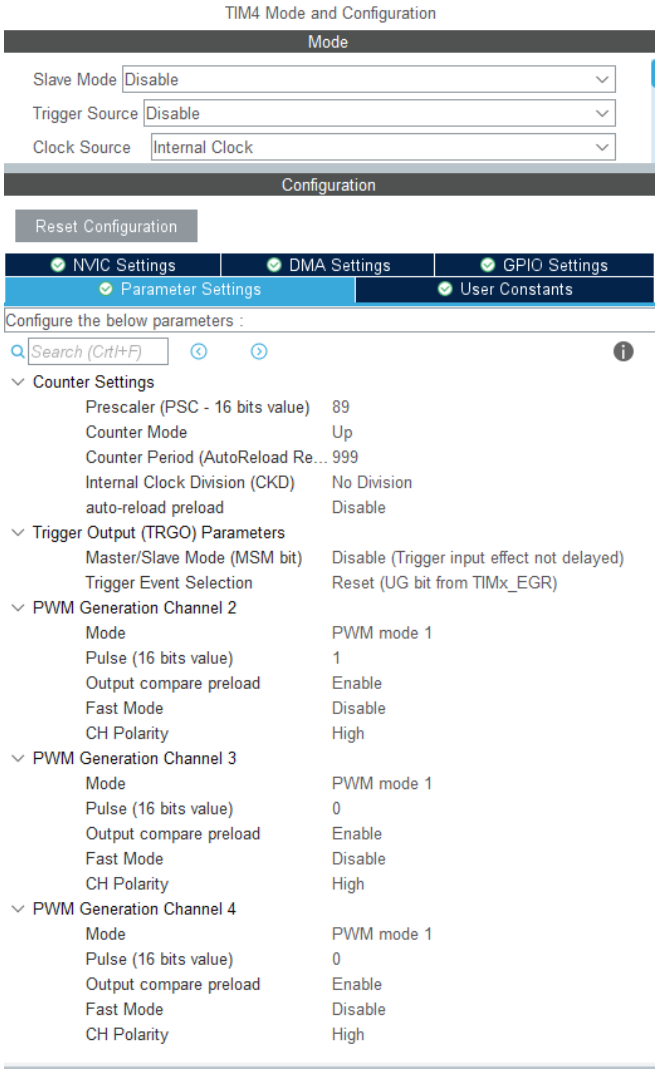
Настройка I2C:



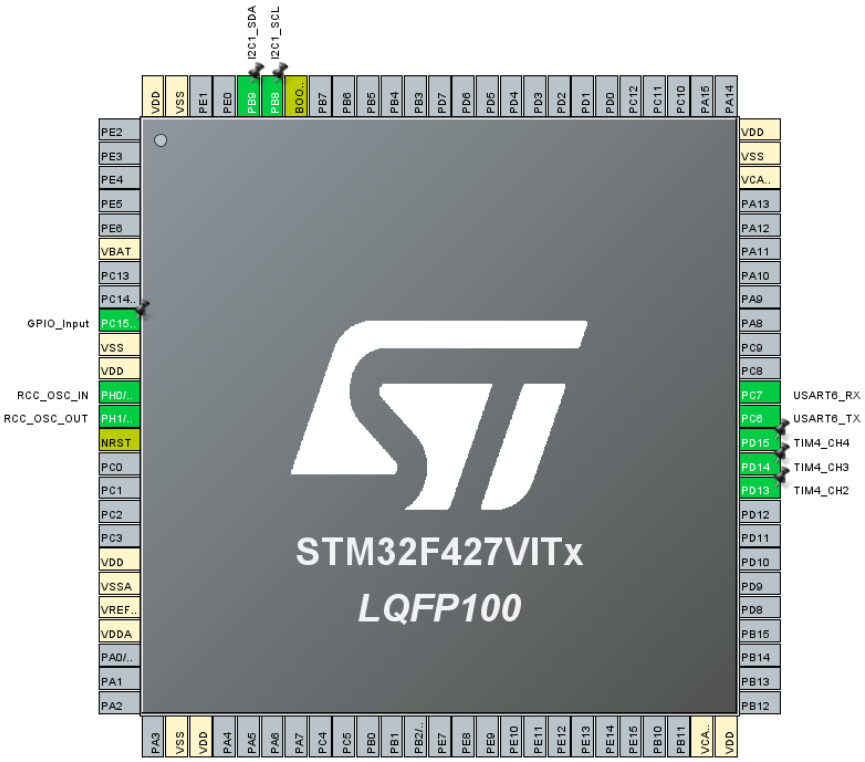
Настройка таймера TIM6:



Настройка таймера TIM4:

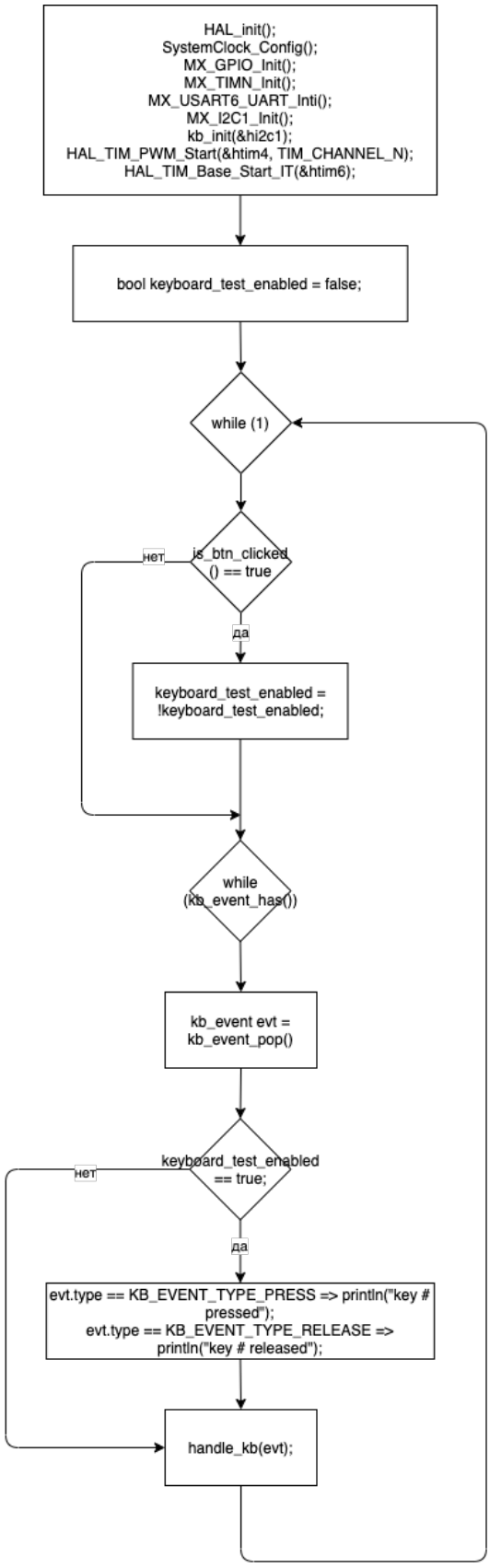


Распиновка:

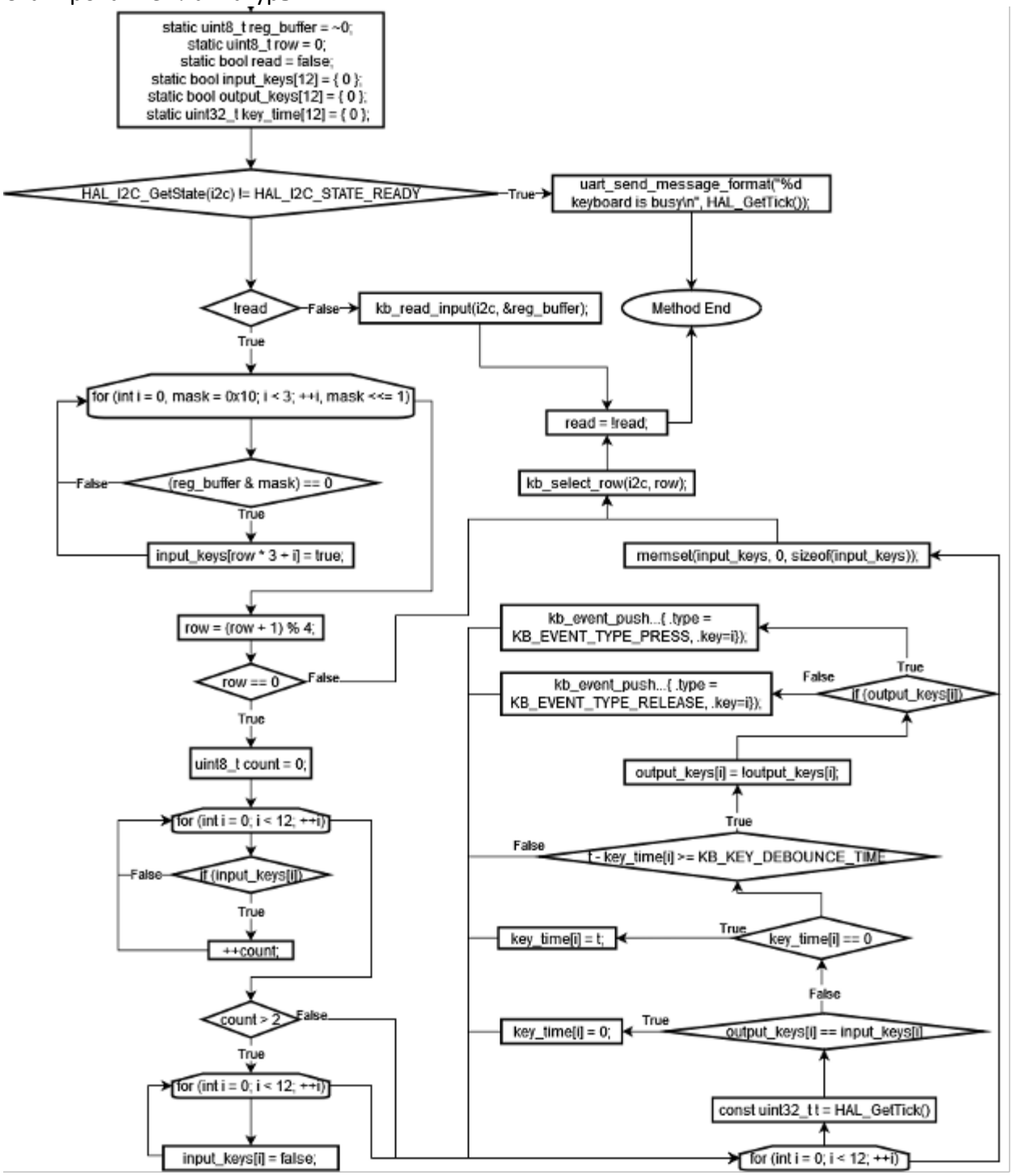

I2C1_SDA
I2C1_SCL
VDD
VSS
PE1
PE0
PB9
PB8
BOO..
PB7
PB6
PB5
PB4
PB3
PD7
PD6
PD5
PD4
PD3
PD2
PD1
PD0
PC12
PC11
PC10
PA15
PA14
PE2
PE3
PE4
PE5
PE6
VBAT
PC13
PC14..
GPIO_Input
PC15..
VSS
VDD
RCC_OSC_IN
PH0/..
RCC_OSC_OUT
PH1/..
NRST
PC0
PC1
PC2
PC3
VDD
VSSA
VREF..
VDDA
PA0/..
PA1
PA2
VDD
VSS
VCA..
PA13
PA12
PA11
PA10
PA9
PA8
PC9
PC8
PC7
USART6_RX
PC6
USART6_TX
PD15
TIM4_CH4
PD14
TIM4_CH3
PD13
TIM4_CH2
PD12
PD11
PD10
PD9
PD8
PB15
PB14
PB13
PB12
PA3
VSS
VDD
PA4
PA5
PA6
PA7
PC4
PC5
PB0
PB1
PB2/..
PE7
PE8
PE9
PE10
PE11
PE12
PE13
PE14
PE15
PB10
PB11
VCA..
VDD
STM32F427VITx
LQFP100

Блок-схемы

Главный программный цикл:



Сканирование клавиатуры:



Код

```
#pragma once

#include <stdint.h>
#include <stdbool.h>

#include "main.h"


enum kb_event_type {
    KB_EVENT_TYPE_PRESS = 0,
    KB_EVENT_TYPE_RELEASE = 1,
};

enum kb_event_key {
    KB_EVENT_KEY_1 = 0,
```

```
    KB_EVENT_KEY_2 = 1,
    KB_EVENT_KEY_3 = 2,
    KB_EVENT_KEY_4 = 3,
    KB_EVENT_KEY_5 = 4,
    KB_EVENT_KEY_6 = 5,
    KB_EVENT_KEY_7 = 6,
    KB_EVENT_KEY_8 = 7,
    KB_EVENT_KEY_9 = 8,
    KB_EVENT_KEY_10 = 9,
    KB_EVENT_KEY_11 = 10,
    KB_EVENT_KEY_12 = 11,
};


struct kb_event {
    enum kb_event_type type;
    enum kb_event_key key;
};


bool kb_event_has();
struct kb_event kb_event_pop();
void kb_init(I2C_HandleTypeDef * i2c);
void kb_scan_step(I2C_HandleTypeDef * i2c);
```

```
#pragma once

#include <stdint.h>


enum LED {
    GREEN = 0,
    YELLOW = 1,
    RED = 2,
};

struct GarlandMode {
    enum LED color;
    uint8_t brightness;
};

extern const char * const led_names[];

void led_set_brightness(enum LED led, uint8_t power);

static void led_mode_enable(struct GarlandMode mode) { led_set_brightness(mode.color,
mode.brightness); }

static void led_mode_disable(struct GarlandMode mode) { led_set_brightness(mode.color, 0); }
```

```
#pragma once

#include <stdint.h>
#include <string.h>


void print(const char * content);
void println(const char * message);
void print_format(const char * format, ...);


#include <KEYBOARD.h>
#include <stdbool.h>
#include <stddef.h>
#include <UART.h>

#define BUFFER_CAPACITY (32)
#define INC_BUFFER_IDX(__idx) do { (__idx) = ((__idx) + 1) % (BUFFER_CAPACITY); } while (0)
#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 (50)
```

```
static struct kb_event buffer[BUFFER_CAPACITY] = { 0 };
static size_t buffer_start_idx = 0;
static size_t buffer_end_idx = 0;


static void kb_event_push(struct kb_event event) {
    buffer[buffer_end_idx] = event;
    INC_BUFFER_IDX(buffer_end_idx);
}

bool kb_event_has() {
       const uint32_t priMask = __get_PRIMASK();
       __disable_irq();
       const bool ret = buffer_start_idx != buffer_end_idx;
       __set_PRIMASK(priMask);
       return ret;
}

struct kb_event kb_event_pop() {
       const uint32_t priMask = __get_PRIMASK();
       __disable_irq();
       const struct kb_event evt = buffer[buffer_start_idx];
       INC_BUFFER_IDX(buffer_start_idx);
       __set_PRIMASK(priMask);
       return evt;
}

void kb_init(I2C_HandleTypeDef * i2c) {
    static uint8_t output = 0x0;
    HAL_I2C_Mem_Write(i2c, KB_I2C_WRITE_ADDRESS, KB_OUTPUT_REG, 1, &output, 1, 100);
}

static void kb_write_config(I2C_HandleTypeDef * i2c, uint8_t data) {
    static uint8_t buf;
    buf = data;
    HAL_I2C_Mem_Write_IT(i2c, KB_I2C_WRITE_ADDRESS, KB_CONFIG_REG, 1, &buf, 1);
}

static void kb_select_row(I2C_HandleTypeDef * i2c, uint8_t row) {
       kb_write_config(i2c, ~((uint8_t) (1 << row)));
}

static void kb_read_input(I2C_HandleTypeDef * i2c, uint8_t * data) {
       HAL_I2C_Mem_Read_IT(i2c, KB_I2C_READ_ADDRESS, KB_INPUT_REG, 1, data, 1); }

void kb_scan_step(I2C_HandleTypeDef * i2c) {
    static uint8_t reg_buffer = ~0;
    static uint8_t row = 0;
    static bool read = false;
    static bool input_keys[12] = { 0 };
    static bool output_keys[12] = { 0 };
    static uint32_t key_time[12] = { 0 };
    if (HAL_I2C_GetState(i2c) != HAL_I2C_STATE_READY) return;
    if (!read) {
        for (uint8_t i = 0, mask = 0x10; i < 3; ++i, mask <<= 1)
            if ((reg_buffer & mask) == 0)
               input_keys[row * 3 + i] = true;
        row = (row + 1) % 4;
        if (row == 0) {
               uint8_t count = 0;
               for (int i = 0; i < 12; ++i)
                      if (input_keys[i])
                             ++count;
               if (count > 2)
                      for (int i = 0; i < 12; ++i)
                             input_keys[i] = false;
            for (int i = 0; i < 12; ++i) {
                const uint32_t t = HAL_GetTick();
                if (output_keys[i] == input_keys[i]) key_time[i] = 0;
                else if (key_time[i] == 0) key_time[i] = t;
                else if (t - key_time[i] >= KB_KEY_DEBOUNCE_TIME) {
                    output_keys[i] = !output_keys[i];
                    if (output_keys[i]) kb_event_push((struct kb_event) { .type =
KB_EVENT_TYPE_PRESS, .key = i });
                    else kb_event_push((struct kb_event) { .type = KB_EVENT_TYPE_RELEASE, .key =
i });
                }
            }
```

```
            memset(input_keys, 0, sizeof(input_keys));
        }
        kb_select_row(i2c, row);
    } else kb_read_input(i2c, &reg_buffer);
    read = !read;
}
```

```
#include <LED.h>
#include "main.h"


extern TIM_HandleTypeDef htim4;

const char * const led_names[] = {
    [GREEN] = "GREEN",
    [YELLOW] = "YELLOW",
    [RED] = "RED",
};


typedef void (* abstract_LED_setter)(uint16_t);

static void set_green_LED(uint16_t power) { htim4.Instance->CCR2 = power; }

static void set_yellow_LED(uint16_t power) { htim4.Instance->CCR3 = power; }

static void set_red_LED(uint16_t power) { htim4.Instance->CCR4 = power; }

static const abstract_LED_setter led_setters[] = {
    [GREEN] = set_green_LED,
    [YELLOW] = set_yellow_LED,
    [RED] = set_red_LED,
};

void led_set_brightness(enum LED led, uint8_t power) {
    if (power > 100) power = 100;
    led_setters[led]((uint16_t) power * 10);
}
```

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2021 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */
/* USER CODE END Header */

/* Includes ------------------------------------------------------------------*/
#include <KEYBOARD.h>
#include <LED.h>
#include <UART.h>
#include "main.h"

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


/* USER CODE END Includes */

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

```c
/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */

#define BTN_DEBOUNCE_TIME (100)

/* USER CODE END PD */

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

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
I2C_HandleTypeDef hi2c1;

TIM_HandleTypeDef htim4;
TIM_HandleTypeDef htim6;

UART_HandleTypeDef huart6;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_TIM4_Init(void);
static void MX_TIM6_Init(void);
static void MX_USART6_UART_Init(void);
static void MX_I2C1_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

static struct GarlandMode led_modes[9] = {
    { .color = GREEN, .brightness = 10 },
    { .color = GREEN, .brightness = 40 },
    { .color = GREEN, .brightness = 100 },
    { .color = YELLOW, .brightness = 10 },
    { .color = YELLOW, .brightness = 40 },
    { .color = YELLOW, .brightness = 100 },
    { .color = RED, .brightness = 10 },
    { .color = RED, .brightness = 40 },
    { .color = RED, .brightness = 100 },
};


static bool is_button_clicked() {
    static bool output_btn = false;
    static uint32_t btn_time = 0;
    const bool input_btn = HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_15) == GPIO_PIN_RESET;
    const uint32_t t = HAL_GetTick();
    if (input_btn == output_btn) btn_time = 0;
    else if (btn_time == 0) btn_time = t;
    else if (btn_time - t >= BTN_DEBOUNCE_TIME) {
        output_btn = !output_btn;
        if (output_btn) return true;
    }
    return false;
}

static bool handle_kb_menu(struct kb_event evt) {
    static struct GarlandMode new_mode = { 0 };
    static uint8_t new_mode_number = 0;
    static bool is_number_selected = false;
    static bool is_color_selected = false;
    static bool may_reset = false;
    switch (evt.key) {
        case KB_EVENT_KEY_1:
        case KB_EVENT_KEY_2:
        case KB_EVENT_KEY_3:
        case KB_EVENT_KEY_4:
```

```
        case KB_EVENT_KEY_5:
        case KB_EVENT_KEY_6:
        case KB_EVENT_KEY_7:
        case KB_EVENT_KEY_8:
        case KB_EVENT_KEY_9:
            if (!is_number_selected) {
                new_mode_number = evt.key;
                is_number_selected = true;
                break;
            }
            switch (evt.key) {
                case KB_EVENT_KEY_1:
                case KB_EVENT_KEY_2:
                case KB_EVENT_KEY_3:
                    is_color_selected = true;
                    new_mode.color = (enum LED) evt.key;
                break;
                case KB_EVENT_KEY_4:
                    if (new_mode.brightness >= 10) new_mode.brightness = new_mode.brightness -
10;
                break;
                case KB_EVENT_KEY_5:
                    new_mode.brightness = (new_mode.brightness + 10) % 110;
                break;
                default:
                    // do nothing
                break;
            }
            break;
        case KB_EVENT_KEY_10:
        case KB_EVENT_KEY_11:
            // do nothing
            break;
        case KB_EVENT_KEY_12:
            if (is_number_selected && is_color_selected) {
                println("new mode saved");
                led_modes[new_mode_number] = new_mode;
            }
            else if (may_reset) println("discarding all changes");
            else {
                may_reset = true;
                if (!is_number_selected) println("you should provide number of new mode");
                if (!is_color_selected) println("you should provide color of new mode");
                println("nothing to save, press again to discard and leave");
                return true;
            }
            println("config mode == OFF");
            return false;
    }
    if (!is_number_selected && !is_color_selected) print_format("new mode params: number is not
selected, color is not selected, brightness = %d%%\r\n", new_mode.brightness);
    else if (!is_number_selected) print_format("new mode params: number is not selected, color =
%s, brightness = %d%%\r\n", led_names[new_mode.color], new_mode.brightness);
    else if (!is_color_selected) print_format("new mode params: number = %d, color is not
selected, brightness = %d%%\r\n", new_mode_number + 1, new_mode.brightness);
    else print_format("new mode params: number = %d, color = %s, brightness = %d%%\r\n",
new_mode_number + 1, led_names[new_mode.color], new_mode.brightness);
    may_reset = false;
    return true;
}

static void handle_kb(struct kb_event evt) {
    static uint8_t current_mode = 9;
    static bool in_menu = false;
    if (evt.type == KB_EVENT_TYPE_PRESS) {
        if (in_menu) in_menu = handle_kb_menu(evt);
        else switch (evt.key) {
               case KB_EVENT_KEY_1:
               case KB_EVENT_KEY_2:
            case KB_EVENT_KEY_3:
            case KB_EVENT_KEY_4:
            case KB_EVENT_KEY_5:
            case KB_EVENT_KEY_6:
            case KB_EVENT_KEY_7:
            case KB_EVENT_KEY_8:
            case KB_EVENT_KEY_9:
            case KB_EVENT_KEY_10:
               if (current_mode < 9) led_mode_disable(led_modes[current_mode]);
```

```
                current_mode = evt.key;
                if (current_mode < 9) {
                        led_mode_enable(led_modes[current_mode]);
                        print_format("activated LED mode #%d\r\n", current_mode + 1);
                } else println("disabled LEDs");
            break;
            case KB_EVENT_KEY_11:
                println("config mode == ON");
                in_menu = true;
            break;
            case KB_EVENT_KEY_12:
                // do nothing
            break;
        }
    }
}

/* 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_TIM4_Init();
  MX_TIM6_Init();
  MX_USART6_UART_Init();
  MX_I2C1_Init();
  /* USER CODE BEGIN 2 */

  kb_init(&hi2c1);
  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);

  /* USER CODE END 2 */



  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  bool keyboard_test_enabled = false;
  while (1) {
    /* USER CODE END WHILE */
        if (is_button_clicked()) {
                if (keyboard_test_enabled) println("keyboard test == ON");
                else println("keyboard test == OFF");
                keyboard_test_enabled = !keyboard_test_enabled;
        }
    /* USER CODE BEGIN 3 */
      while (kb_event_has()) {
         struct kb_event evt = kb_event_pop();
         if (keyboard_test_enabled) {
```

```
                switch (evt.type) {
                        case KB_EVENT_TYPE_PRESS:
                                print_format("key %d pressed\r\n", evt.key + 1);
                        break;
                        case KB_EVENT_TYPE_RELEASE:
                             print_format("key %d released\r\n", evt.key + 1);
                        break;
                }
            } else handle_kb(evt);
        }
  }  /* 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 CPU, AHB and APB busses clocks
  */
  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 busses 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();
  }
}

/**
  * @brief I2C1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C1_Init(void)
{

  /* USER CODE BEGIN I2C1_Init 0 */

  /* USER CODE END I2C1_Init 0 */

  /* USER CODE BEGIN I2C1_Init 1 */

  /* USER CODE END I2C1_Init 1 */
  hi2c1.Instance = I2C1;
  hi2c1.Init.ClockSpeed = 400000;
```

```
  hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure Analogue filter
  */
  if (HAL_I2CEx_ConfigAnalogFilter(&hi2c1, I2C_ANALOGFILTER_ENABLE) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure Digital filter
  */
  if (HAL_I2CEx_ConfigDigitalFilter(&hi2c1, 0) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C1_Init 2 */

  /* USER CODE END I2C1_Init 2 */

}

/**
  * @brief TIM4 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM4_Init(void)
{

  /* USER CODE BEGIN TIM4_Init 0 */

  /* USER CODE END TIM4_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};

  /* USER CODE BEGIN TIM4_Init 1 */

  /* USER CODE END TIM4_Init 1 */
  htim4.Instance = TIM4;
  htim4.Init.Prescaler = 89;
  htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim4.Init.Period = 999;
  htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim4.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim4) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim4, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_Init(&htim4) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_PWM_ConfigChannel(&htim4, &sConfigOC, TIM_CHANNEL_2) != HAL_OK)
```

```
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_ConfigChannel(&htim4, &sConfigOC, TIM_CHANNEL_3) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_ConfigChannel(&htim4, &sConfigOC, TIM_CHANNEL_4) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM4_Init 2 */

  /* USER CODE END TIM4_Init 2 */
  HAL_TIM_MspPostInit(&htim4);

}

/**
  * @brief TIM6 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM6_Init(void)
{

  /* USER CODE BEGIN TIM6_Init 0 */

  /* USER CODE END TIM6_Init 0 */

  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM6_Init 1 */

  /* USER CODE END TIM6_Init 1 */
  htim6.Instance = TIM6;
  htim6.Init.Prescaler = 8999;
  htim6.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim6.Init.Period = 9;
  htim6.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim6) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim6, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM6_Init 2 */

  /* USER CODE END TIM6_Init 2 */

}

/**
  * @brief USART6 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART6_UART_Init(void)
{

  /* USER CODE BEGIN USART6_Init 0 */

  /* USER CODE END USART6_Init 0 */

  /* USER CODE BEGIN USART6_Init 1 */

  /* USER CODE END USART6_Init 1 */
  huart6.Instance = USART6;
  huart6.Init.BaudRate = 115200;
  huart6.Init.WordLength = UART_WORDLENGTH_8B;
  huart6.Init.StopBits = UART_STOPBITS_1;
  huart6.Init.Parity = UART_PARITY_NONE;
  huart6.Init.Mode = UART_MODE_TX_RX;
  huart6.Init.HwFlowCtl = UART_HWCONTROL_NONE;
```

```
  huart6.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart6) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART6_Init 2 */

  /* USER CODE END USART6_Init 2 */

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin : PC15 */
  GPIO_InitStruct.Pin = GPIO_PIN_15;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef * htim) {
    if (htim->Instance == TIM6) {
        kb_scan_step(&hi2c1);
    }
}

/* 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 */

  /* 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,
     tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```

```c
#include <stdbool.h>
#include <stdarg.h>
#include <stdio.h>
#include <UART.h>

#include "main.h"


extern UART_HandleTypeDef huart6;


static inline bool uart_is_ready() { return HAL_UART_GetState(&huart6) == HAL_UART_STATE_READY; }

void print(const char * content) {
	while (!uart_is_ready());
	HAL_UART_Transmit_IT(&huart6, (void *) content, strlen(content));
}

void println(const char * message) {
	print(message);
	print("\r\n");
}

void print_format(const char * format, ...) {
	static char buffer[1024];
	while (!uart_is_ready());
	va_list ap;
	va_start(ap, format);
	vsnprintf(buffer, sizeof(buffer), format, ap);
	va_end(ap);
	print(buffer);
}
```

Выводы

В процессе выполнения лабораторной работы мы научились работать с клавиатурным блоком стенда SDK-1.1M, считывать нажатие и отпускание клавиш с него. Научились пользоваться таймерами стенда и с их помощью имитировать яркость светодиодов.