

Факультет программной̆ инженерии и компьютерной̆ техники

Проектирование вычислительных систем

Лабораторная работа №3: Таймеры и интерфейс I2C   
Вариант 2

Преподаватель: Пинкевич Василий Юрьевич

Выполнили: Тарасов Александр Станиславович, Кульбако Артемий Юрьевич Р34112

# 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.

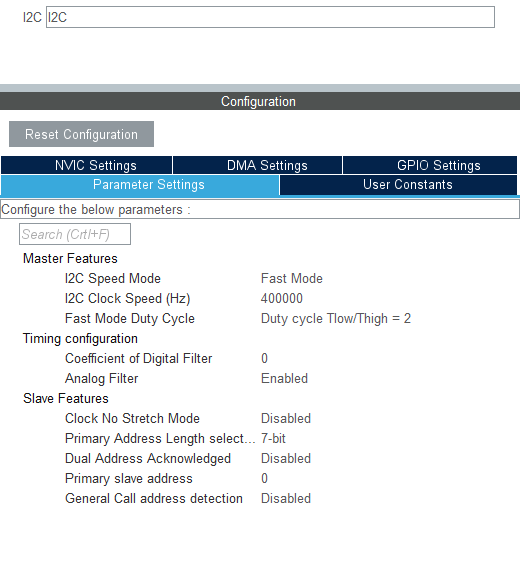
После входе в меню настройки сначала надо нажать кнопку, привязанный к которой режим требуется изменить, далее кнопками 1 – 3 выбирается светодиод (зеленый, желтый, красный) и кнопками 4, 5 – коэффициент заполнения от 0 до 100% с шагом 10%. По нажатию кнопки выхода из меню новый режим сохраняется.

# Настройка

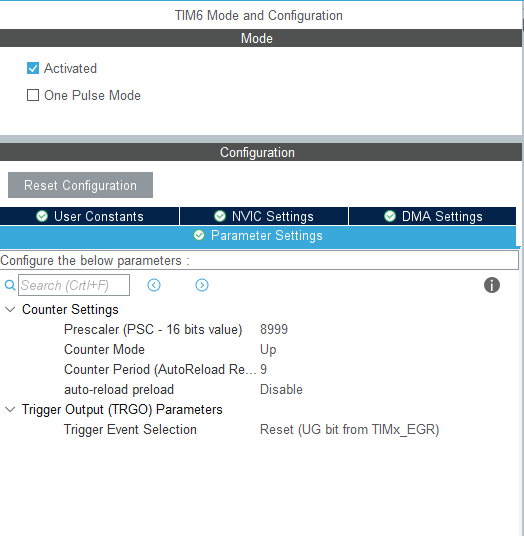
Микросхема PCA9538 является внешним расширителем GPIO-портов ввода/вывода, который подключается к микроконтроллеру по интерфейсу I2C. Данная микросхема реализует подключение до 8 сигналов, каждый из которых может быть настроен как вход или выход.

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.

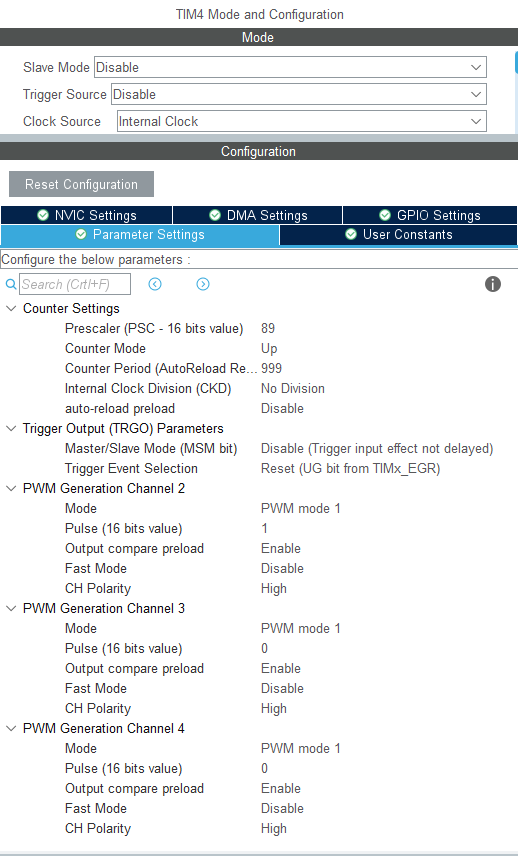
Настройка I2C:

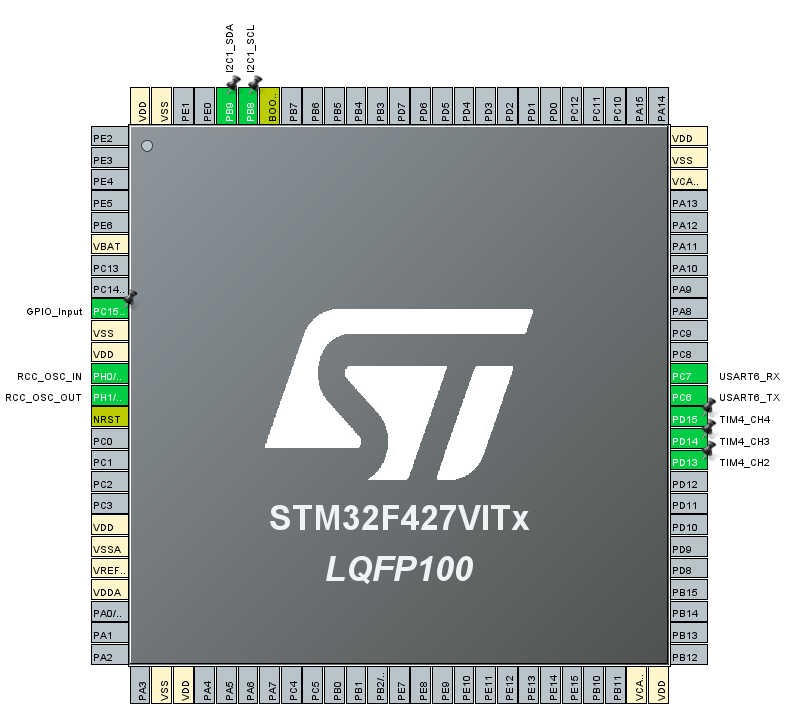


Настройка таймера TIM6:

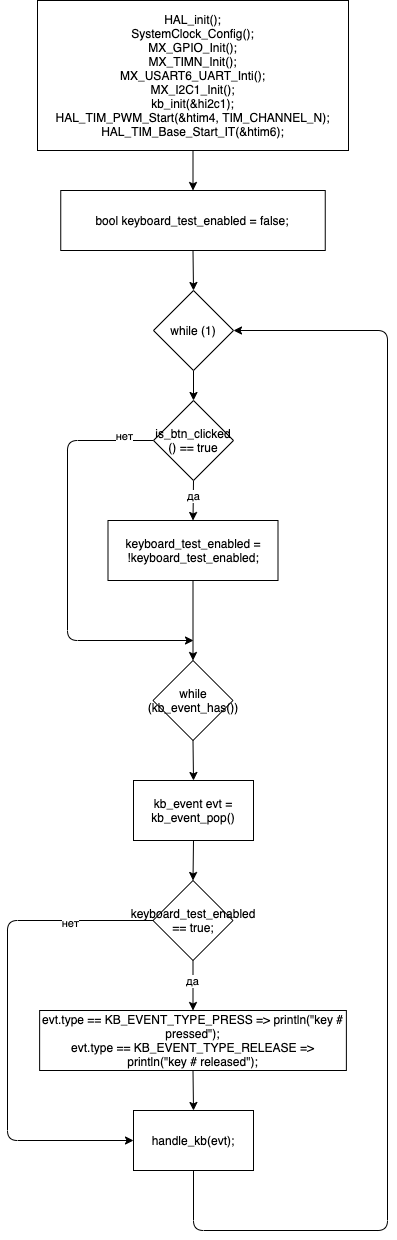


Настройка таймера TIM4:

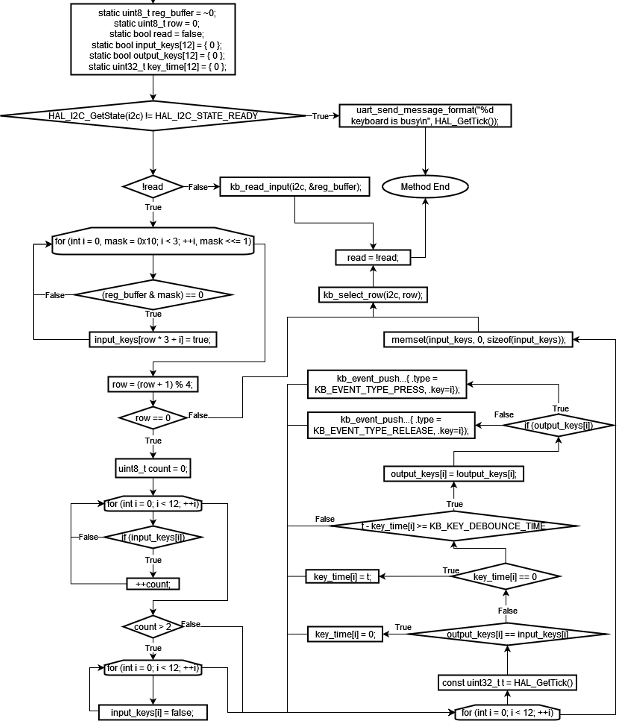


Распиновка:

# 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 \*/

/\* 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\*\*\*\*/

**#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.