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

Дисциплина: Проектирование вычислительных систем

Лабораторная работа 4

Вариант 2

Выполнили:
Колбасин Владислав Ильич,
Митичев Иван Дмитриевич,
группа P3416

Преподаватель:
Пинкевич Василий Юрьевич

2025 г.
Санкт-Петербург

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 2:

Реализовать настраиваемый пульт управления светодиодами боковой панели. Пульт должен иметь два основных режима: рабочий режим и режим настройки.

Действия стенда при получении символов от компьютера в рабочем режиме:

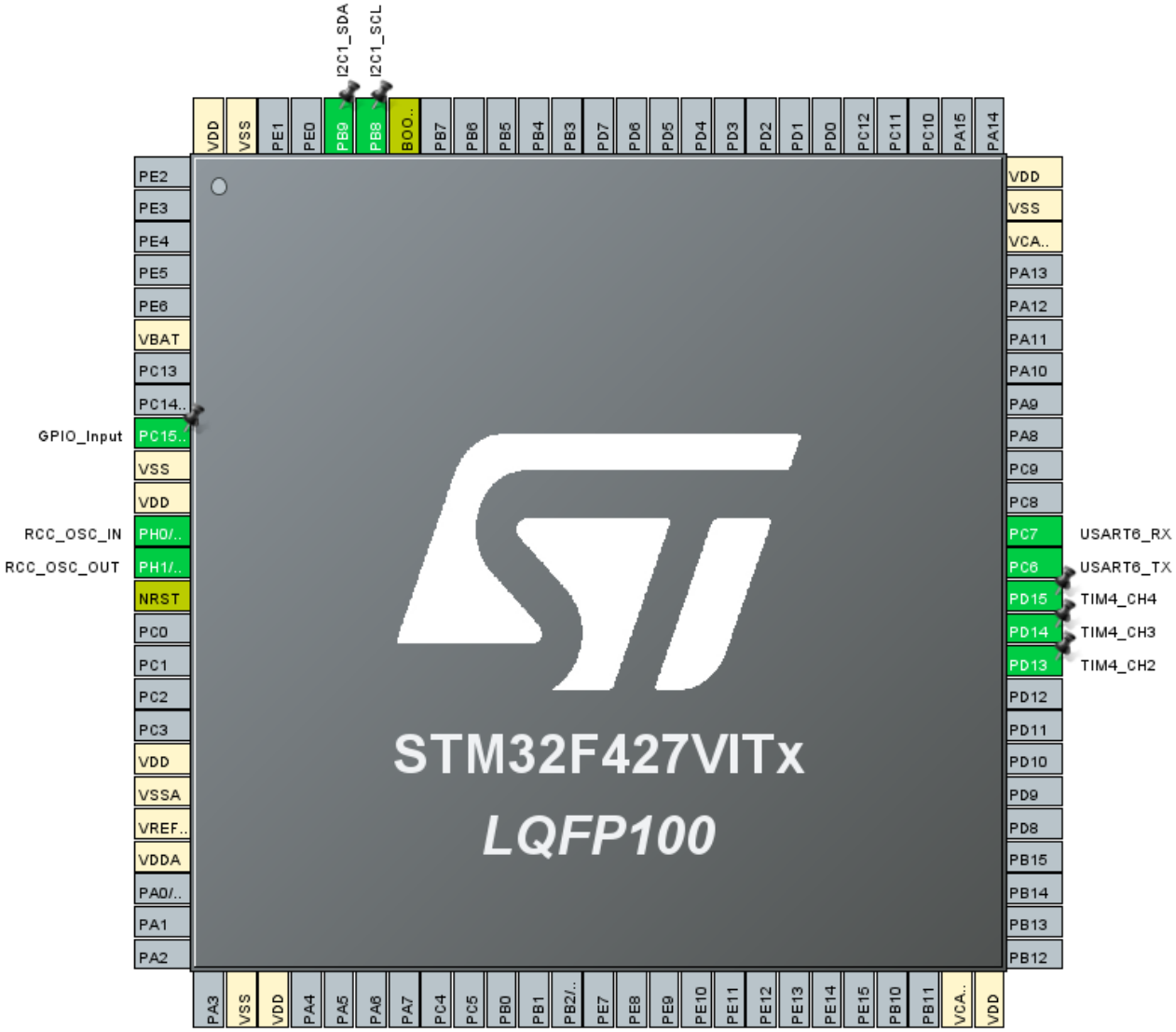
Символ	Действие
«1» – «9»	Зажигание светодиода в соответствии с настройками пульта. Светодиод горит до ввода следующего символа. Предыдущий светодиод выключается. Настройки по умолчанию: «1» – зеленый, 10 % яркости; «2» – зеленый, 40 % яркости; «3» – зеленый, 100 % яркости; «4» – желтый, 10 % яркости; «5» – желтый, 40 % яркости; «6» – желтый, 100 % яркости; «7» – красный, 10 % яркости; «8» – красный, 40 % яркости; «9» – красный, 100 % яркости.
«0»	Погасить все светодиоды.
«Enter»	Вход в режим настройки.

После входа в меню настройки сначала надо ввести символ от «1» до «9», настройки для которого требуется изменить. Далее символом «a», «b» или «c» выбирается светодиод (зеленый, желтый, красный соответственно). После этого несколькими нажатиями кнопок «+» и «-» задается коэффициент заполнения от 0 до 100 % с шагом 10 %. По нажатию кнопки «Enter» сохраняется новая настройка и происходит переход в рабочий режим.

По вводу каждого символа в UART должно выводиться сообщение о том, какой режим активирован, или текущие вводимые настройки.

Частота процессорного ядра – 45 МГц, частота ШИМ-сигнала для светодиодов – 250 Гц.

Конфигурация пинов



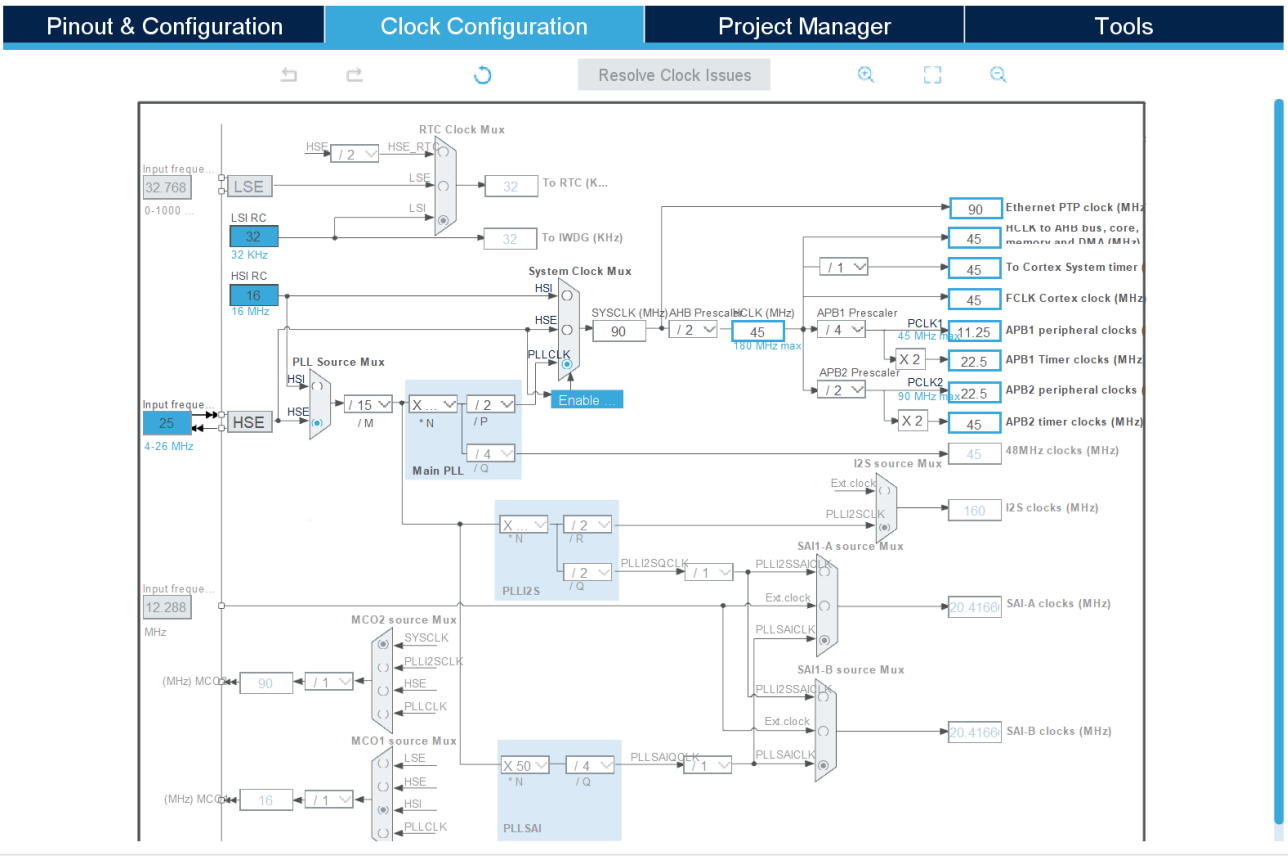
Описание контактов

- **PB9** – I2C_SDA
- **PB8** –I2C_SCL
- **PH0** – RSS_OSC_IN
- **PH1** – RSS_OSC_OUT
- **PD13** – TIM4_CH2 (зеленый светодиод)
- **PD14** – TIM4_CH3 (желтый светодиод)
- **PD15** – TIM4_CH4 (красный светодиод)
- **PC6** – USART6_TX
- **PC7** – USART6_RX

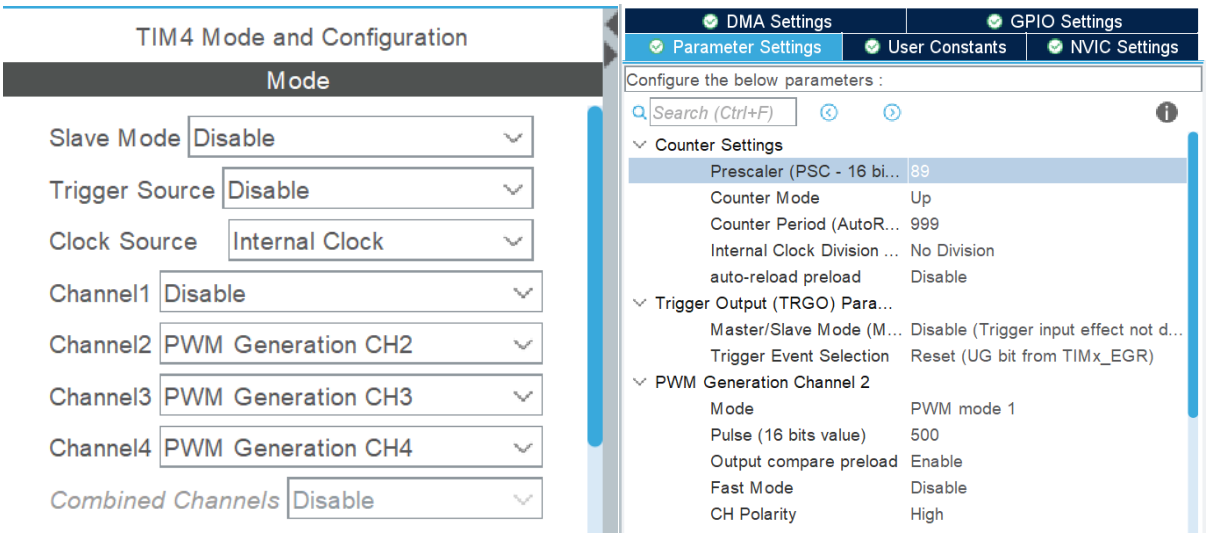
- **PC15** – GPIO_Input

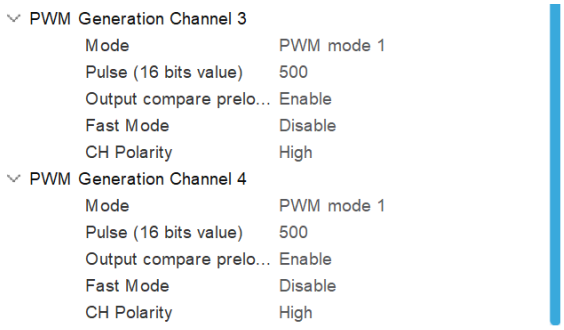
Конфигурация таймера повторяет конфигурацию из 3й лабораторной работы.

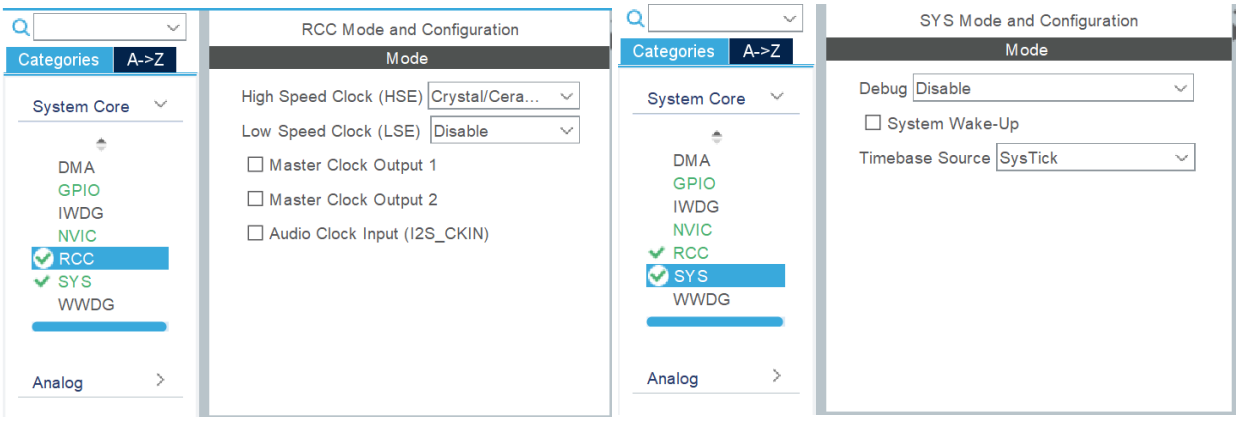
По условию задания необходимо выставить частоту процессорного ядра 45 МГц. Тогда по шине APB1 частота таймера будет 22.5 МГц (в программе используется TIM4).



Рассчитаем значения для прескейлера и AutoReload Register. По условию частота ШИМ-сигнала для светодиодов должна быть 250 Гц. Значит нужно уменьшить частоту таймера в 22500000/250 = 90000 раз. Выберем прескейлер 89 и ARR = 999:

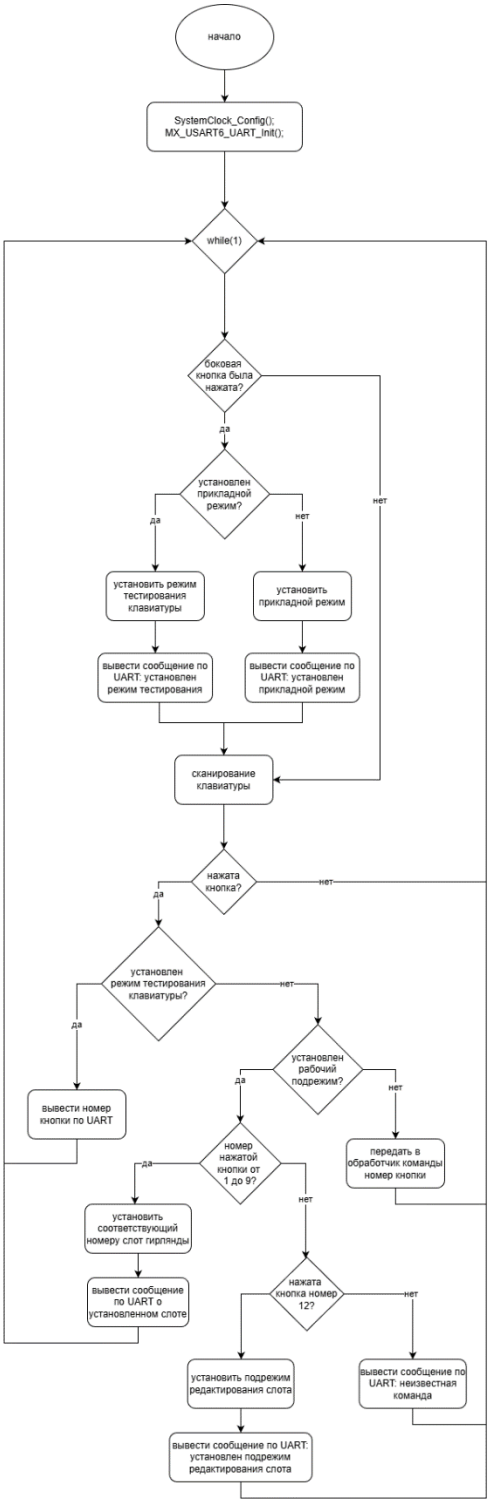

TIM4 Mode and Configuration
Mode
Slave Mode Disable
Trigger Source Disable
Clock Source Internal Clock
Channel1 Disable
Channel2 PWM Generation CH2
Channel3 PWM Generation CH3
Channel4 PWM Generation CH4
Combined Channels Disable
DMA Settings
GPIO Settings
Parameter Settings
User Constants
NVIC Settings
Configure the below parameters :
Search (Ctrl+F)
Counter Settings
Prescaler (PSC - 16 bi... 89
Counter Mode Up
Counter Period (AutoR... 999
Internal Clock Division ... No Division
auto-reload preload Disable
Trigger Output (TRGO) Para...
Master/Slave Mode (M... Disable (Trigger input effect not d...
Trigger Event Selection Reset (UG bit from TIMx_EGR)
PWM Generation Channel 2
Mode PWM mode 1
Pulse (16 bits value) 500
Output compare preload Enable
Fast Mode Disable
CH Polarity High


PWM Generation Channel 3
Mode PWM mode 1
Pulse (16 bits value) 500
Output compare prelo... Enable
Fast Mode Disable
CH Polarity High
PWM Generation Channel 4
Mode PWM mode 1
Pulse (16 bits value) 500
Output compare prelo... Enable
Fast Mode Disable
CH Polarity High

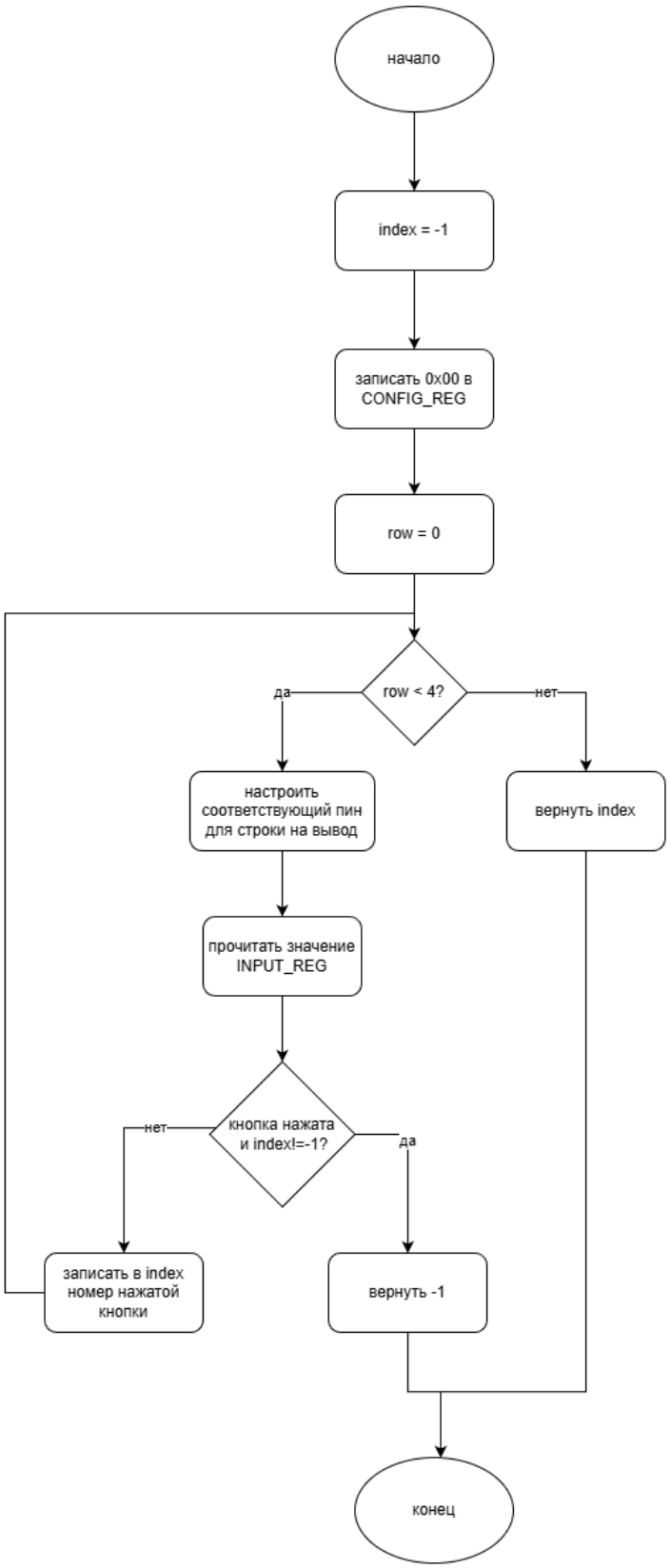

Categories
A->Z
System Core
DMA
GPIO
IWDG
NVIC
RCC
SYS
WWDG
Analog
RCC Mode and Configuration
Mode
High Speed Clock (HSE) Crystal/Cera...
Low Speed Clock (LSE) Disable
Master Clock Output 1
Master Clock Output 2
Audio Clock Input (I2S_CKIN)
Categories
A->Z
System Core
DMA
GPIO
IWDG
NVIC
RCC
SYS
WWDG
Analog
SYS Mode and Configuration
Mode
Debug Disable
System Wake-Up
Timebase Source SysTick

Блок-схемы

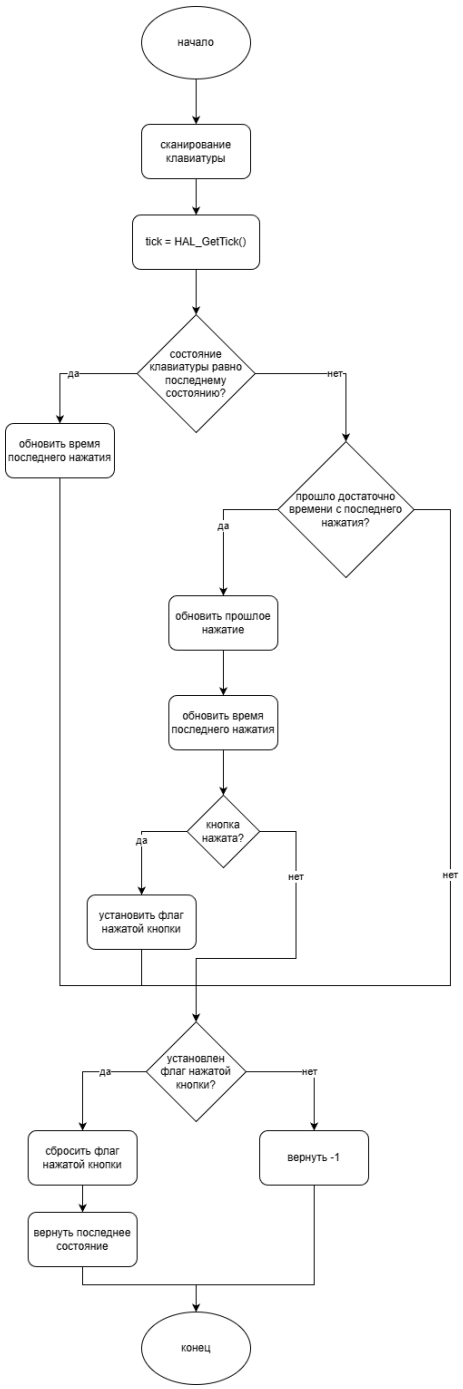
Основная программа:



Подпрограмма сканирования клавиатуры:



Подпрограмма защиты от двойных нажатий и дребезга:



Код программы

main.c:

```
/* USER CODE BEGIN Header */
/**
 ******************************************************************************
 * @file           : main.c
 * @brief          : Main program body
 ******************************************************************************
 * @attention
 *
 * Copyright (c) 2025 STMicroelectronics.
 * All rights reserved.
 *
 * This software is licensed under terms that can be found in the LICENSE file
 * in the root directory of this software component.
 * If no LICENSE file comes with this software, it is provided AS-IS.
 *
 ******************************************************************************
 */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "i2c.h"
#include "tim.h"
#include "usart.h"
#include "gpio.h"
#include "button_driver.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "usart_drv.h"
#include <stdio.h>
#include <stdint.h>
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
typedef enum
{
  PRACTICAL_MODE,
  KEYBOARD_TEST_MODE
} CurrentMode_t;
CurrentMode_t current_mode = PRACTICAL_MODE;

typedef struct {
    int last_state;
    uint32_t last_tick;
    bool pressed_flag;
} KB_Button_t;

typedef enum
{
  MODE_WORK,
  MODE_CONFIG_SLOT,
```

```
  MODE_CONFIG_COLOR,
  MODE_CONFIG_BRIGHTNESS
} Mode_t;
Mode_t mode = MODE_WORK;

typedef enum
{
  LED_GREEN,
  LED_YELLOW,
  LED_RED
} LedColor_t;

typedef struct
{
  LedColor_t color;
  uint8_t duty;
} LedSlot_t;

LedSlot_t slots[9] = {
    {LED_GREEN, 10}, {LED_GREEN, 40}, {LED_GREEN, 100}, {LED_YELLOW, 10}, {LED_YELLOW,
40}, {LED_YELLOW, 100}, {LED_RED, 10}, {LED_RED, 40}, {LED_RED, 100}};

uint8_t selected_slot = 0;
/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define KB_ADDR (0x71 << 1)

#define BUF_SIZE 8192

#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_DEBOUNCE_DELAY_MS (50)
#define KB_MAX_KEYS 12
/* USER CODE END PD */

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

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
/* USER CODE BEGIN PFP */
void uart_println(const char *s)
{
```

```
  USART_DRV_TxStr(s);
  USART_DRV_TxStr("\n");
}
void set_led_pwm(LedSlot_t cfg)
{
  __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_2, 0);
  __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_3, 0);
  __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_4, 0);

  uint32_t value = cfg.duty * 10;

  switch (cfg.color)
  {
  case LED_GREEN:
    __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_2, value);
    break;
  case LED_YELLOW:
    __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_3, value);
    break;
  case LED_RED:
    __HAL_TIM_SET_COMPARE(&htim4, TIM_CHANNEL_4, value);
    break;
  }
}

static uint8_t edit_slot = 0;

void handle_input(int input)
{
  if ((mode == MODE_CONFIG_SLOT) && (input >= 1 && input <= 9))
  {
    edit_slot = input - 1;
    set_led_pwm(slots[edit_slot]);
    mode = MODE_CONFIG_COLOR;
    uart_println("Select LED color: 1(green), 2(yellow), 3(red)");
  }
  else if ((mode == MODE_CONFIG_COLOR) && (input == 1 || input == 2 || input == 3))
  {
    switch (input)
    {
    case 1:
      slots[edit_slot].color = LED_GREEN;
      break;
    case 2:
      slots[edit_slot].color = LED_YELLOW;
      break;
    case 3:
      slots[edit_slot].color = LED_RED;
      break;
    }
    set_led_pwm(slots[edit_slot]);
    mode = MODE_CONFIG_BRIGHTNESS;
    uart_println("Use +/- to change brightness, Enter to save");
  }
  else if ((mode == MODE_CONFIG_BRIGHTNESS) && (input == 1 || input == 2))
  {
    int delta = (input == 1) ? 10 : -10;
```

```
    int result = slots[edit_slot].duty + delta;
    if (result > 100)
    {
      result = 100;
    }
    else if (result < 0)
    {
      result = 0;
    }
    slots[edit_slot].duty = result;
    set_led_pwm(slots[edit_slot]);

    char buf[32];
    sprintf(buf, "Brightness: %d%%", slots[edit_slot].duty);
    uart_println(buf);
  }
  else if (input == 12)
  {
    mode = MODE_WORK;
    uart_println("Saved, back to WORK mode");
  }
  else
  {
    uart_println("Invalid input");
  }
}

static KB_Button_t kb_button = { .last_state = -1, .last_tick = 0, .pressed_flag = false
};
int KB_ScanRaw(void)
{
    uint8_t reg_buffer = 0xFF;
    uint8_t tmp = 0x00;
    int i = 0;
    int index = -1;

    HAL_I2C_Mem_Write(&hi2c1, KB_I2C_WRITE_ADDRESS, KB_OUTPUT_REG, 1, &tmp, 1, 100);

    for (int row = 0; row < 4; row++)
    {
        uint8_t cfg = ~(1 << row);
        HAL_I2C_Mem_Write(&hi2c1, KB_I2C_WRITE_ADDRESS, KB_CONFIG_REG, 1, &cfg, 1, 100);
        HAL_Delay(5);
        HAL_I2C_Mem_Read(&hi2c1, KB_I2C_READ_ADDRESS, KB_INPUT_REG, 1, &reg_buffer, 1,
100);

        uint8_t colbits = reg_buffer >> 4;

        if (colbits != 7 && index != -1)
            return -1;

        switch (colbits)
        {
            case 6:
             index = (i==0)? row*3 + 1 : -1;
             i++;
             break;
```

```
            case 5:
             index = (i==0)? row*3 + 2 : -1;
             i++;
             break;
            case 3:
             index = (i==0)? row*3 + 3 : -1;
             i++;
             break;
            default: break;
        }
    }

    return index;
}

int KB_Scan(void)
{
    int raw = KB_ScanRaw();
    uint32_t tick = HAL_GetTick();

    if (raw != kb_button.last_state)
    {
        if ((tick - kb_button.last_tick) >= KB_DEBOUNCE_DELAY_MS)
        {
            kb_button.last_state = raw;
            kb_button.last_tick = tick;
            if (raw != -1)
            {
                kb_button.pressed_flag = true;
            }
        }
    }
    else
    {
        kb_button.last_tick = tick;
    }

    if (kb_button.pressed_flag)
    {
        kb_button.pressed_flag = false;
        return kb_button.last_state;
    }
    return -1;
}

/* USER CODE END PFP */

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

/* USER CODE END 0 */

/**
 * @brief  The application entry point.
 * @retval int
 */
int main(void)
```

```c
{
  /* 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_I2C1_Init();
  MX_TIM4_Init();
  MX_USART6_UART_Init();
  /* USER CODE BEGIN 2 */
  USART_DRV_Init(&huart6);
  USART_DRV_EnableInterrupts(true);
  USART_DRV_SetEcho(true);

  HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_2);
  HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_3);
  HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_4);
  set_led_pwm(slots[selected_slot]);

//  uint8_t config = 0b11110000;
//  HAL_I2C_Mem_Write(&hi2c1, KB_ADDR, 0x03, 1, &config, 1, 100);
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  int input = -1;
  while (1)
  {
    BUTTON_Process();

    if (BUTTON_WasPressed())
    {
      if (current_mode == PRACTICAL_MODE)
      {
        current_mode = KEYBOARD_TEST_MODE;
        uart_println("SET KEYBOARD_TEST_MODE");
      }
      else
      {
        current_mode = PRACTICAL_MODE;
```

```
        uart_println("SET PRACTICAL_MODE");
      }
    }

    input = KB_Scan();

    if (input != -1)
    {
      if (current_mode == KEYBOARD_TEST_MODE)
      {
        char tmp_buf[16];
        sprintf(tmp_buf, "%u", input);
        uart_println(tmp_buf);
      }
      else
      {
        if (mode != MODE_WORK)
        {
          handle_input(input);
          continue;
        }
        switch (input)
        {
        case 1 ... 9:
          set_led_pwm(slots[input - 1]);
          char buf[32];
          sprintf(buf, "Set mode %d", input);
          uart_println(buf);
          break;
        case 12:
          mode = MODE_CONFIG_SLOT;
          USART_DRV_TxStr("CONFIG_MODE\nSelect slot (from 1 to 9):\n");
          break;
        default:
          uart_println("UNKNOWN COMMAND");
          break;
        }
      }
    }

    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* 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_SCALE3);

  /** Initializes the RCC Oscillators according to the specified parameters
   * in the RCC_OscInitTypeDef structure.
   */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 15;
  RCC_OscInitStruct.PLL.PLLN = 108;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 4;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
   */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK | RCC_CLOCKTYPE_SYSCLK |
RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV2;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
  {
    Error_Handler();
  }
}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
 * @brief  This function is executed in case of error occurrence.
 * @retval None
 */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef USE_FULL_ASSERT
/**
 * @brief  Reports the name of the source file and the source line number
```

```
 *         where the assert_param error has occurred.
 * @param  file: pointer to the source file name
 * @param  line: assert_param error line source number
 * @retval None
 */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

Вывод

В ходе выполнения лабораторной работы была разработана программа для работы с матричной клавиатурой через интерфейс I²C. Реализована подсистема опроса клавиатуры с защитой от дребезга, обеспечивающая корректное фиксирование одиночных и множественных нажатий кнопок без повторов, с присвоением кодов каждой кнопке. Программа поддерживает два режима работы — тестовый, где коды нажатых кнопок выводятся в UART, и прикладной, в котором нажатия обрабатываются в соответствии с заданием, с переключением режимов по кнопке на боковой панели стенд.