Факультет программной инженерии и компьютерной техники

Лабораторная работа №4

«Интерфейс I2C и матричная клавиатура»

по дисциплине «Проектирование вычислительных систем»

Вариант – 1

Выполнили:

Студенты группы P3432

Чмурова М.В.

Комягин Д.А.

Преподаватель:

Пинкевич Василий Юрьевич

Санкт-Петербург

2025

Задание

Разработать программу, которая использует интерфейс I 2C 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 задания:

Задания аналогичны вариантам лабораторной работы №3, за исключением того, что ввод символов должен выполняться не с клавиатуры через UART, а с помощью клавиатуры стенда. Выбор кнопок клавиатуры стенда, играющих роль кнопок клавиатуры компьютера должен выполняться по усмотрению исполнителей. В отчете необходимо привести описание функций кнопок в реализованной программе.

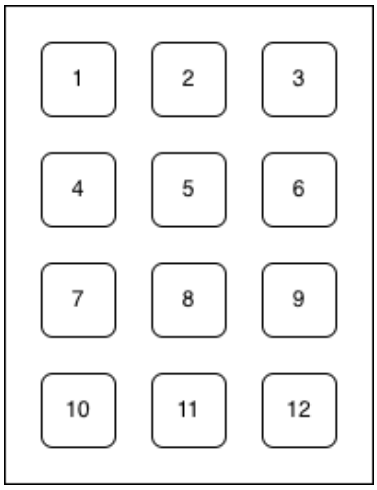

Рисунок 1. Схематичное представление кнопок матричной клавиатуры SDK-1.1M

Кнопка	Действие
1-7	Воспроизведение одной ноты (от «до» до «си») текущей октавы с текущей длительностью звучания. Начальные значения: первая октава (пятая по порядку), длительность 1 с
8	Увеличение номера текущей октавы (максимальная – пятая).
9	Уменьшение номера текущей октавы (минимальная – субконтроктава).
10	Увеличение длительности воспроизведения ноты на 0,1 с (максимум – 5 с).
11	Уменьшение длительности воспроизведения ноты на 0,1 с (минимум – 0,1 с).
12	Последовательное воспроизведение всех нот текущей октавы с текущей длительностью без пауз.

Описание организации программы и структуры драйверов

Для использования микросхемы-расширителя портов PCA9538 были использованы функции чтения/записи регистров с использованием функций *HAL_I2C_Mem_Write()* и *HAL_I2C_Mem_Read()*:

```
HAL_StatusTypeDef PCA9538_Read_Register(uint16_t addr, pca9538_register reg,
uint8_t* buf)
{
	return HAL_I2C_Mem_Read(&hi2c1, addr, reg, 1, buf, 1, 100);
}

HAL_StatusTypeDef PCA9538_Write_Register(uint16_t addr, pca9538_register reg,
uint8_t* buf)
{
	return HAL_I2C_Mem_Write(&hi2c1, addr, reg, 1, buf, 1, 100);
}
```

Кроме того, использован *enum* хранящий названия регистров расширителя, которые были использованы в программе:

```
typedef enum{
    INPUT_PORT = 0x00,
    OUTPUT_PORT = 0x01,
    POLARITY_INVERSION = 0x02,
    CONFIG = 0x03
} pca9538_register;
```

А также отдельные функции для чтения из INPUT_PORT и записи в регистры CONFIG, POLARITY_INVERSION и OUTPUT_PORT:

```
HAL_StatusTypeDef PCA9538_Read_Inputs(uint16_t addr, uint8_t* buf)
{
	return PCA9538_Read_Register(addr, INPUT_PORT, buf);
}

HAL_StatusTypeDef PCA9538_Write_Config(uint16_t addr, uint8_t* buf)
{
	return PCA9538_Write_Register(addr, CONFIG, buf);
}

HAL_StatusTypeDef PCA9538_Write_Polarity(uint16_t addr, uint8_t* buf)
{
	return PCA9538_Write_Register(addr, POLARITY_INVERSION, buf);
}

HAL_StatusTypeDef PCA9538_Write_Output(uint16_t addr, uint8_t* buf)
{
	return PCA9538_Write_Register(addr, OUTPUT_PORT, buf);
}
```

Для использования матричной клавиатуры перед запуском происходит ее инициализация:

```
HAL_StatusTypeDef keyboard_init(void)
{
    uint8_t zero = 0;

    if (PCA9538_Write_Polarity(KBRD_ADDR, &zero) != HAL_OK)
        return HAL_ERROR;

    if (PCA9538_Write_Output(KBRD_ADDR, &zero) != HAL_OK)
        return HAL_ERROR;

    return HAL_OK;
}
```

Инверсия входов не используется (устанавливается в 0), выходные порты при инициализации устанавливаются в ноль (изменения будут происходить динамически в *keyboard_get_raw()*.

Тогда соответственно функция для получения «сырого» значения с матричной клавиатуры:

```
uint8_t keyboard_get_raw(void)
{
    uint8_t input_port;
    uint8_t detected_row = 0xFF;
    uint8_t detected_column = 0xFF;
    uint8_t press_count = 0;

    for (uint8_t row = 0; row < ROWS; row++)
    {
        uint8_t config_row = ROW_MASKS[row];
        PCA9538_Write_Config(KBRD_ADDR, &config_row);

        uint32_t t0 = HAL_GetTick();
        while (HAL_GetTick() - t0 < 1) {}

        PCA9538_Read_Inputs(KBRD_ADDR, &input_port);
        uint8_t pushed_columns = input_port & 0x70;
        if (pushed_columns != 0x70)
        {
            for (uint8_t column = 0; column < COLUMNS; column++)
            {
                if (!(pushed_columns & COLUMN_MASKS[column]))
                {
                    press_count++;
                    if (press_count > 1)
                        return 0;
                    detected_row = row;
                    detected_column = column;
                }
            }
        }
    }
    if (press_count == 1)
        return detected_row * 3 + detected_column + 1;

    return 0;
}
```

Каждый столбец (P0-P3) устанавливаются в цикле на выход со значением 0. Затем происходит небольшая задежрка (1 мс.), чтобы значения успели установиться, после получаются входные порты в переменную input_port, и все порты кроме P4-P6 зануляются. В цикле для столбцов определяется номер столбца, получившего нажатие, и возвращается номер найденной кнопки.

Если кнопок было нажато несколько, то возвращается 0, так как множественное/одновременное нажатие кнопок запрещено.

Для обработки ситуации дребезга кнопки реализована функция:

```
uint8_t keyboard_update(void)
{
    uint8_t raw = keyboard_get_raw();

    if (raw != last_raw)
    {
        last_raw = raw;
        change_time = HAL_GetTick();
        return 0;
    }
    if (HAL_GetTick() - change_time < DEBOUNCE_TIME_MS)
        return 0;

    if (last_raw != stable_key)
    {
        stable_key = last_raw;
        return stable_key;
    }

    return 0;
}
```

В ней используются переменные, изначально установленные в ноль:

- *stable_key*: последняя полученная стабильная кнопка
- *last_raw*: последнее мгновенное «сырое» нажатие
- *change_time*: момент изменения *raw*

В ходе работы функции определяется есть ли изменение состояния *raw* и сохранение, если произошли изменения (нажатие). Если в течение *DEBOUNCE_TIME_MS* (10 мс.) изменений в значении кнопки не произошло, то значение *stable_key* меняется на новое.

Для изменения режимов работы программы (тестовый и прикладной) реализована функция:

```
void handle_operating_mode(void)
{
	uint32_t current_time = HAL_GetTick();

	if (current_time - last_button_update >= 10)
	{
		last_button_update = current_time;
		if (button_is_pressed(&switch_mode_button)) {
			current_button_mode = (current_button_mode == TEST_MODE ?
APPLICATION_MODE : TEST_MODE);

			char msg[64];
			sprintf(msg, "Button pressed. Current mode: %s",
current_button_mode == TEST_MODE ? "TEST" : "APPLICATION");
			uart_transmitln_string(msg);
		}
	}

	if (current_button_mode == TEST_MODE)
		key_handler();
	if (current_button_mode == APPLICATION_MODE)
		notes_handler();
}
```

Здесь по нажатию кнопки изменяется состояние глобальной переменной *current_button_mode* и в зависимости от этого состояния вызываются *key_handler()* для обработки тестового режима:

```
void key_handler(void)
{
	uint8_t key = keyboard_update();
	if (key != 0)
	{
		char msg[32];
		sprintf(msg, "Key=%u", key);
		uart_transmitln_string("");
		uart_transmitln_string(msg);
	}
}
```

А также *notes_handler()* для обработки прикладного режима:

```
void notes_handler(void)
{
	uint8_t key = keyboard_update();
	if (key != 0)
	{
		notes_handle_key(key);
	}

	tone_update();
	play_harmony_update();
}
```

Функции использованные в *notes_handler()* аналогичны функциям использованных в Лабораторной работе №3

Блок-схема прикладного алгоритма

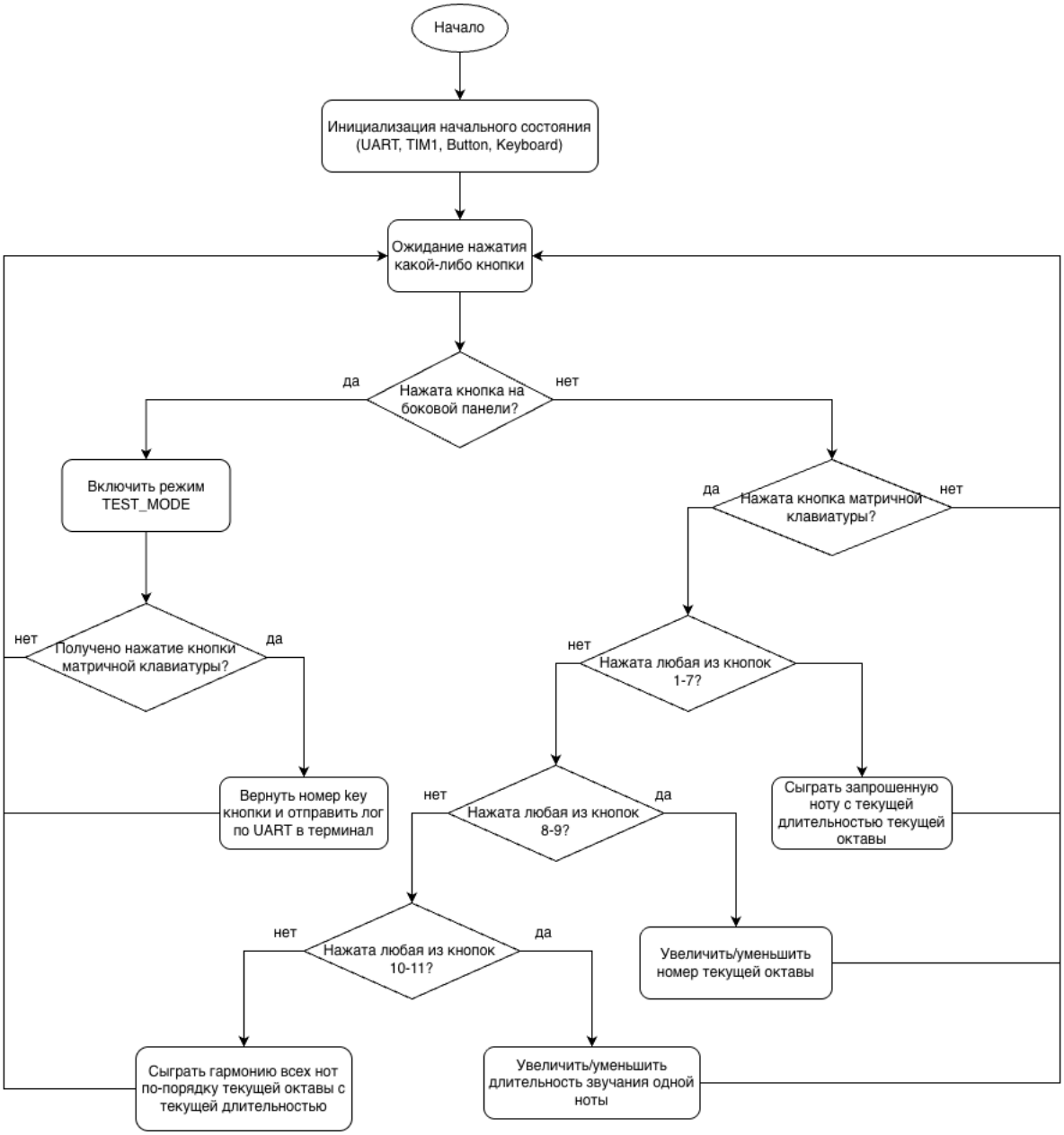

Рисунок 2. Блок-схема алгоритма

Исходный код

В последующем разделе приведён разбор только исходных файлов с расширением *.c*. Заголовочные файлы *.h* не рассматриваются, поскольку не содержат полезной информации, описания каких-либо структур или же они были рассмотрены выше.

pca9538_driver.c

```
#include "drivers/pca9538_driver.h"


HAL_StatusTypeDef PCA9538_Read_Register(uint16_t addr, pca9538_register reg,
uint8_t* buf)
{
	return HAL_I2C_Mem_Read(&hi2c1, addr, reg, 1, buf, 1, 100);
}

HAL_StatusTypeDef PCA9538_Write_Register(uint16_t addr, pca9538_register reg,
uint8_t* buf)
{
	return HAL_I2C_Mem_Write(&hi2c1, addr, reg, 1, buf, 1, 100);
}

HAL_StatusTypeDef PCA9538_Read_Inputs(uint16_t addr, uint8_t* buf)
{
	return PCA9538_Read_Register(addr, INPUT_PORT, buf);
}

HAL_StatusTypeDef PCA9538_Write_Config(uint16_t addr, uint8_t* buf)
{
	return PCA9538_Write_Register(addr, CONFIG, buf);
}

HAL_StatusTypeDef PCA9538_Write_Polarity(uint16_t addr, uint8_t* buf)
{
	return PCA9538_Write_Register(addr, POLARITY_INVERSION, buf);
}

HAL_StatusTypeDef PCA9538_Write_Output(uint16_t addr, uint8_t* buf)
{
	return PCA9538_Write_Register(addr, OUTPUT_PORT, buf);
}
```

keyboard_driver.c:

```
#include "drivers/keyboard_driver.h"

static uint8_t stable_key = 0;
static uint8_t last_raw = 0;
static uint32_t change_time = 0;

static const uint8_t ROW_MASKS[4] = {0xFE, 0xFD, 0xFB, 0xF7};
static const uint8_t COLUMN_MASKS[3] = {0x10, 0x20, 0x40};

HAL_StatusTypeDef keyboard_init(void)
{
```

```
    uint8_t zero = 0;

    if (PCA9538_Write_Polarity(KBRD_ADDR, &zero) != HAL_OK)
        return HAL_ERROR;

    if (PCA9538_Write_Output(KBRD_ADDR, &zero) != HAL_OK)
        return HAL_ERROR;

    return HAL_OK;
}


uint8_t keyboard_get_raw(void)
{
    uint8_t input_port;
    uint8_t detected_row = 0xFF;
    uint8_t detected_column = 0xFF;
    uint8_t press_count = 0;

    for (uint8_t row = 0; row < ROWS; row++)
    {
        uint8_t config_row = ROW_MASKS[row];
        PCA9538_Write_Config(KBRD_ADDR, &config_row);

        uint32_t t0 = HAL_GetTick();
        while (HAL_GetTick() - t0 < 1) {}

        PCA9538_Read_Inputs(KBRD_ADDR, &input_port);
        uint8_t pushed_columns = input_port & 0x70;
        if (pushed_columns != 0x70)
        {
            for (uint8_t column = 0; column < COLUMNS; column++)
            {
                if (!(pushed_columns & COLUMN_MASKS[column]))
                {
                    press_count++;
                    if (press_count > 1)
                        return 0;
                    detected_row = row;
                    detected_column = column;
                }
            }
        }
    }
    if (press_count == 1)
        return detected_row * 3 + detected_column + 1;

    return 0;
}

uint8_t keyboard_update(void)
{
    uint8_t raw = keyboard_get_raw();

    if (raw != last_raw)
    {
        last_raw = raw;
        change_time = HAL_GetTick();
        return 0;
    }
    if (HAL_GetTick() - change_time < DEBOUNCE_TIME_MS)
```

```
        return 0;

    if (last_raw != stable_key)
    {
        stable_key = last_raw;
        return stable_key;
    }

    return 0;
}
```

operating_mode.c:

```
#include "music/operating_mode.h"

button_t switch_mode_button;
uint32_t last_button_update = 0;
button_work_mode_t current_button_mode = APPLICATION_MODE;

void project_init(void) {
      HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);

      if (keyboard_init() != HAL_OK)
            uart_transmitln_string("Keyboard init FAILED");

      button_init(&switch_mode_button, BUTTON_PORT, GPIO_PIN_15);
}

void handle_operating_mode(void)
{
      uint32_t current_time = HAL_GetTick();

      if (current_time - last_button_update >= 10)
      {
            last_button_update = current_time;
            if (button_is_pressed(&switch_mode_button)) {
                  current_button_mode = (current_button_mode == TEST_MODE ?
APPLICATION_MODE : TEST_MODE);

                  char msg[64];
                  sprintf(msg, "Button pressed. Current mode: %s",
current_button_mode == TEST_MODE ? "TEST" : "APPLICATION");
                  uart_transmitln_string(msg);
            }
      }

      if (current_button_mode == TEST_MODE)
            key_handler();
      if (current_button_mode == APPLICATION_MODE)
            notes_handler();
}

void key_handler(void)
{
      uint8_t key = keyboard_update();
      if (key != 0)
      {
            char msg[32];
            sprintf(msg, "Key=%u", key);
            uart_transmitln_string("");
            uart_transmitln_string(msg);
      }
```

```
}
```

С полной версией программы можно ознакомиться, перейдя на GitHub-репозиторий, расположенный по ссылке: https://github.com/kkettch/design-of-computing-systems-semester-7

Вывод

В ходе выполнения работы были успешно разработано и реализовано ПО для управления звуком «пианино», который позволяет воспроизводить ноты с переключением октав во всем рабочем диапазоне. Для их воспроизведения используется матричная клавиатура, подключённая к портам расширителя PCA9538 по I2C. PCA9538 предоставляет 4 регистра: INPUT, OUTPUT, POLARITY_INVERSION, CONFIG. А также передаются сообщения с МК на ПК для вывода диагностических сообщений с использованием интерфейса UART.

Кроме того, реализована возможность изменения длительности звучания ноты в диапазоне от 0.1 до 5 секунд.