МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ
И ПРОГРАММНОЙ ИНЖЕНЕРИИ (КАФЕДРА №43)

КУРСОВАЯ РАБОТА (ПРОЕКТ)
ЗАЩИЩЕНА С ОЦЕНКОЙ

РУКОВОДИТЕЛЬ:

| Доцент, к.т.н. | / | / | / | А. А. Попов |
|---|---|---|---|---|
| (должность, учёная степень, звание) | | (подпись) | (дата защиты) | (инициалы, фамилия) |

# ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
# К КУРСОВОИ РАБОТЕ (ПРОЕКТУ)

## Калькулятор

## ПО КУРСУ: «ПРОГРАММИРОВАНИЕ ВСТРОЕННЫХ ПРИЛОЖЕНИЙ»

РАБОТУ ВЫПОЛНИЛ (А) СТУДЕНТ (КА) ГРУППЫ: 4936 / Нестеренко М.Ю.
Зинатов Р. Г.

(инициалы, фамилия)

/ /

(подпись студента) (дата отчета)

Санкт-Петербург 2022

Содержание

1. Задание на курсовое проектирование

Реализовать многострочный калькулятор на основе Keypad 4x4 и LCD 320x240. Вводимые символы: '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '.', '=', '+', '-', '*', '/'.

Текущую историю хранить в памяти и после включения питания восстанавливать. Точность типа float.

Обязательно реализовать контроль корректности ввода. Не указанные ограничения и форматы доопределить самостоятельно.

На экране отображать виртуальную клавиатуру и подсвечивать на ней нажимаемые кнопки:

| | | | |
|---|---|---|---|
| 7 | 8 | 9 | + |
| 4 | 5 | 6 | - |
| 1 | 2 | 3 | * |
| 0 | . | = | / |

## 2. Техническое задание на прибор

В рамках разрабатываемой программе «Калькулятор» выделяются следующие исполнители (актеры):

– Пользователь;

– Микроконтроллер (как управляющее устройство).

Основные функции калькулятора:

- В режиме бездействия, когда пользователь не взаимодействует с программой, на LCD дисплее демонстрируется интерфейс со всеми основными функциями программы;
- Ввод пользователем следующих полей:
    1) Очистка экрана («AC»);
    2) Цифры в диапазоне [0; 9];
    3) Арифметическая операция («+», «-», «*», «/»);
    4) Символы, запускающие вычисления («+», «-», «*», «/», «=») при записанном в буфере втором числе;
    5) Операции, записывающие числа в память микроконтроллера («MRC», «M+», «M-»).
- Вычисление значения по введенным полям;
- Запись клиентских чисел в память, их чтение и операции «+», «-» над ними.

## 2.1 Диаграмма вариантов использования

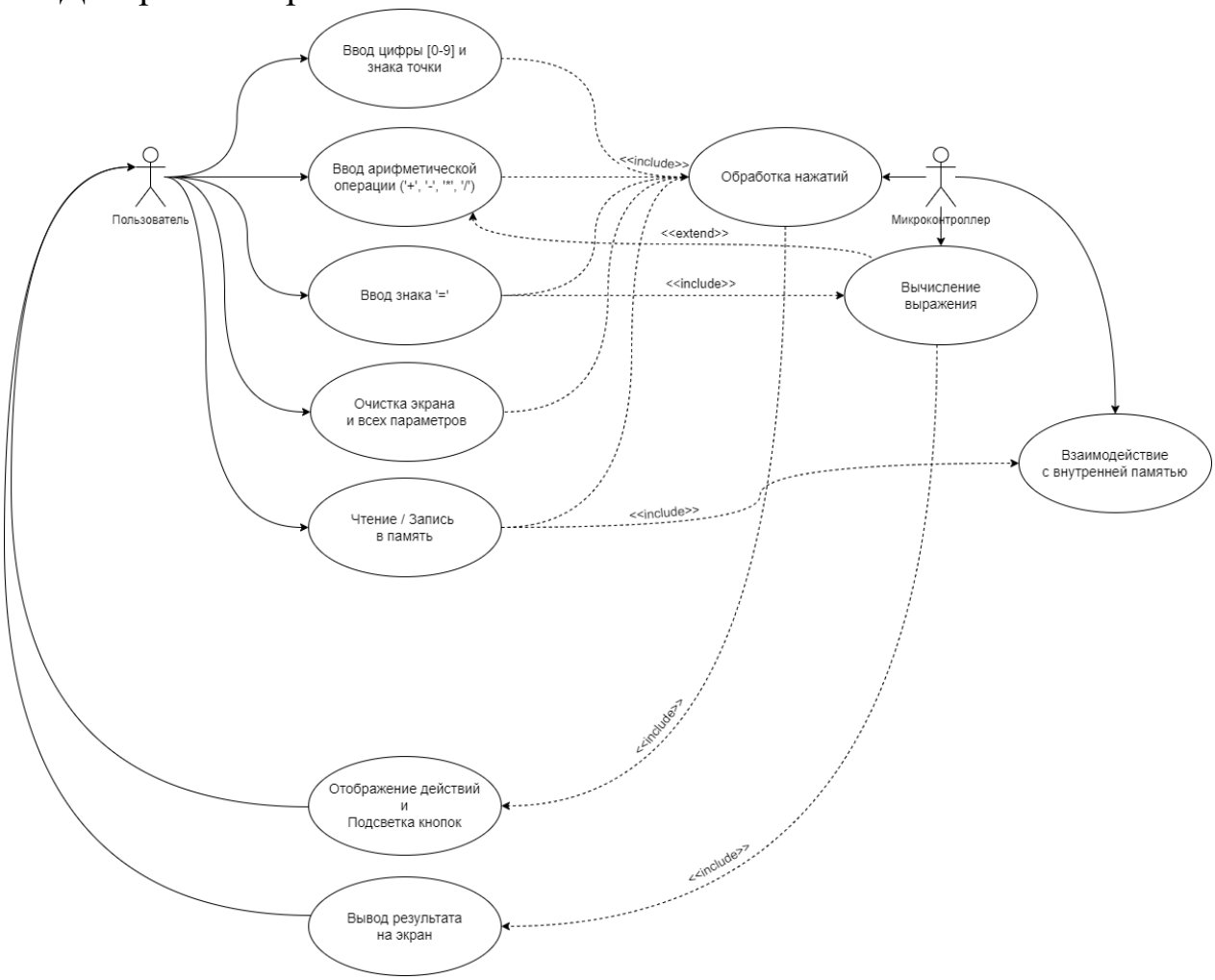


Рисунок 1. Диаграмма вариантов использования

## 2.2 Основные требования

- Входы

  Touch Screen LCD 320x240;

- Выходы

  LCD 320x240;

- Функции

  1) Вычисление основных арифметических операций над числами с плавающий запятой типа float;
  2) Для считываний нажатий используется Touch Screen;

3) Чтение / Запись числа в память и возможность оперировать им при вычислениях;
4) Наличие режимов работы:
   «Серия равенств» (повторение последней операции при неоднократном нажатии клавиши «=» на дисплее) и
   «Серия операций» (при непрерывном вводе операций без ввода знака равенства для подтверждения вычисления

- Особенности
  1) Сенсорный экран Touch Screen LCD 320x240;
  2) Возможность производить операции над числами с плавающей запятой;

- Питание
  Батарейки 3В или от сети переменного тока через стандартный блок питания (USB адаптер);

- Размеры и вес
  Достаточно маленькие, чтобы поместиться в карман;

- Стоимость производства
  Отладочная плата STM32F103C6 - 82 руб.;
  TFT ЖК-экран сенсорный экран ILI9325 – 176 руб.;
  Сборка и тестирование - 50 рублей.

В проекте используется дисплей с разрешением 320 на 240 пикселей, построенный на базе контроллера ILI9325. Он оснащен графической памятью размером 172820 байт с 18-ю битами на пиксель, блоком управляющих регистров и рядом интерфейсов для взаимодействия с микроконтроллером. Из

всех интерфейсов на макете реализован только 16-ти битный интерфейс i80. Схематично взаимодействие с контроллером представлено на рисунке 2.

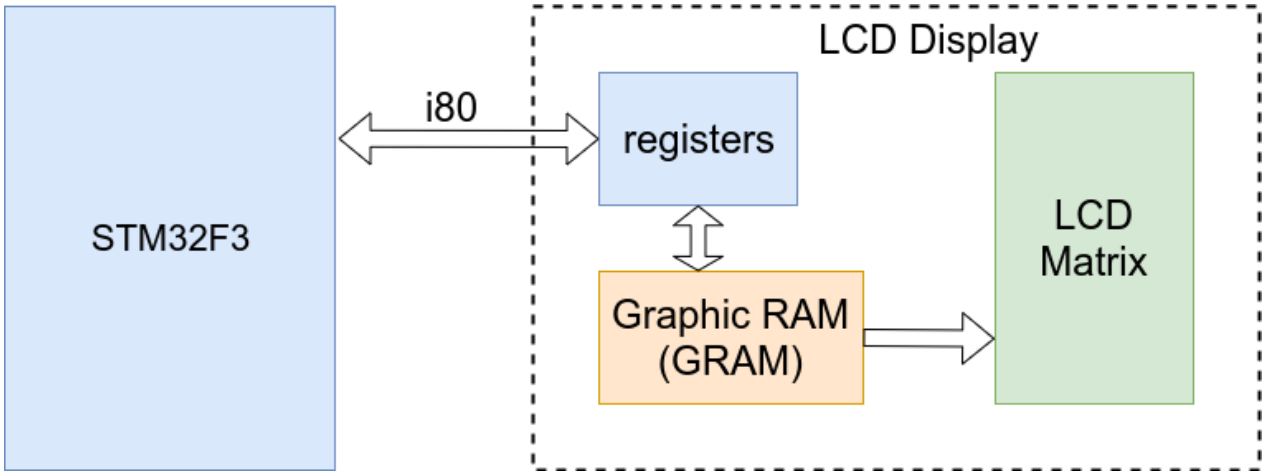


Рисунок 2. Взаимодействие STM32 с LCD дисплеем

## 3. Схемы и настройка проекта

Используемый сенсорный экран является резистивным и имеет следующее устройство:

Резистивные панели имеют многослойную структуру, состоящую из двух проводящих поверхностей, разделенных специальным изолирующим составом, распределенным по всей площади активной области экрана (рисунок 3). При касании наружного слоя, выполненного из тонкого прозрачного пластика, его внутренняя проводящая поверхность совмещается с проводящим слоем основной пластины, благодаря чему происходит изменение сопротивления всей системы. Это изменение фиксируется контроллером панели.

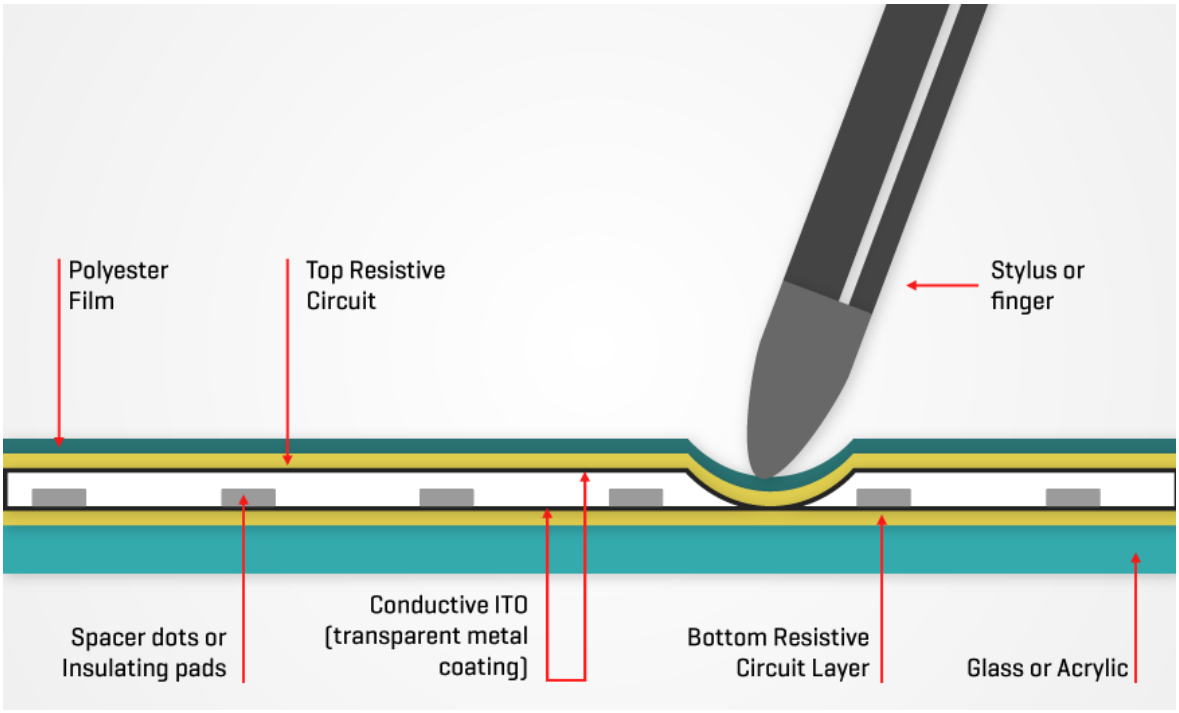


Рисунок 3. Устройство резистивной сенсорной панели

Также, для построения интерактивного пользовательского интерфейса была выбрана графическая библиотека LVGL (Light and Versatile Graphics Library). Библиотека поддерживает дисплеи разного типа (монохромные и цветные), разнообразные устройства ввода (клавиатура, мышь, сенсорный экран) и имеет поддержку сразу нескольких дисплеев.

Подробную информацию можно найти на сайте библиотеки и github:

- https://lvgl.io/
- https://github.com/lvgl/lvgl

Ввиду отсутствия программных стандартов интерфейсов дисплеев и устройств ввода, обычно для подключения библиотеки необходимо самостоятельно реализовать:

- функцию передачи изображения от библиотеки на экран;
- функцию считывания текущего положения курсора.

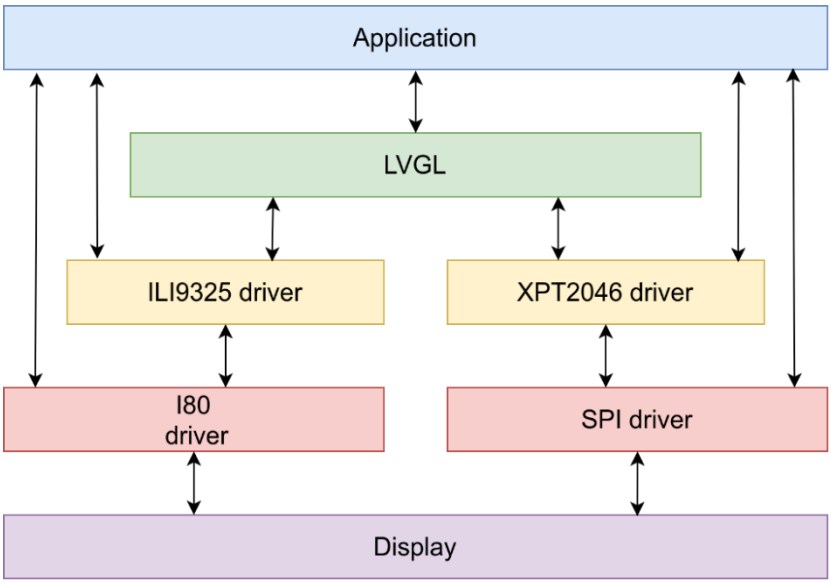


Рисунок 4. Структурная схема подключения дисплея к МК и программе с использованием LVGL

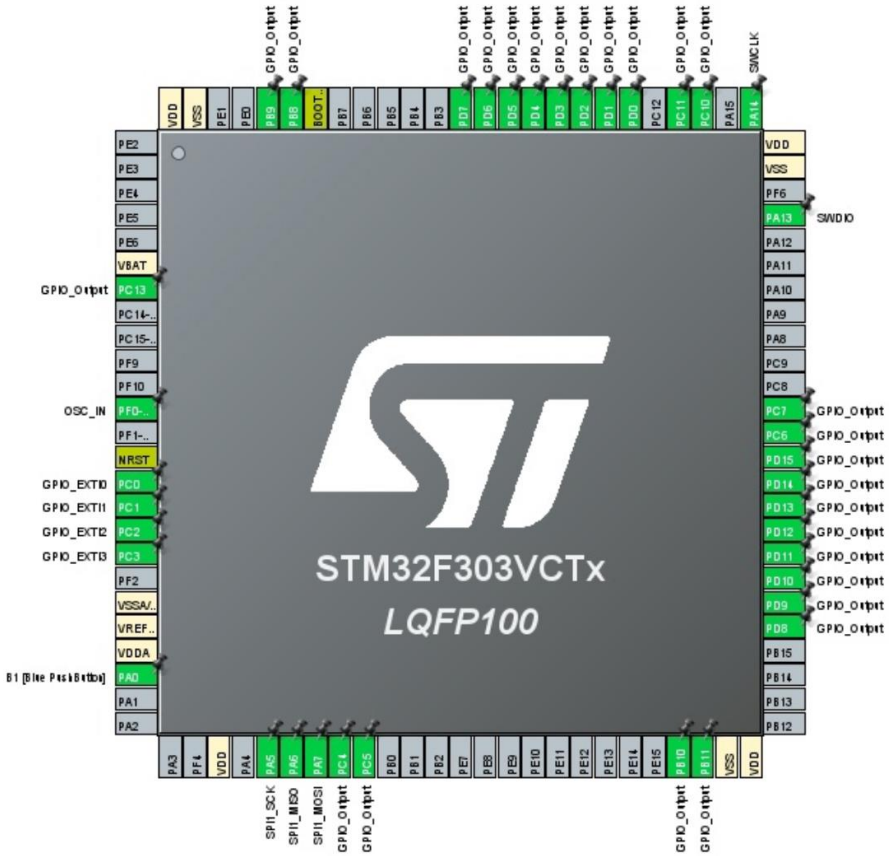


Рисунок 5 - Настройка пинов в программе STM32CubeMX

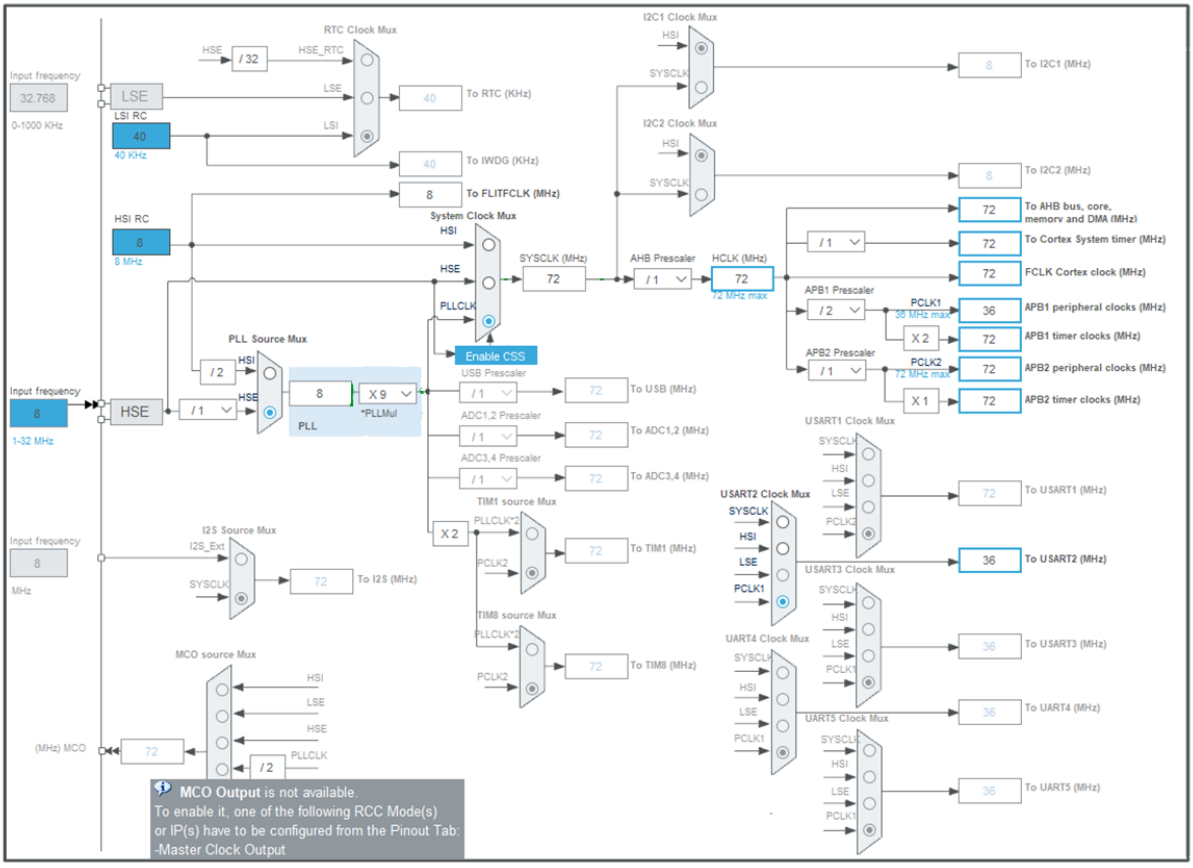


Рисунок 6 - Настройка тактирования в программе STM32CubeMX

Для возможности сохранять и читать числа из памяти был использован регистр резервного копирования, который использует RTC, так как этот регистр является периферийной частью этого модуля. Подключим его.

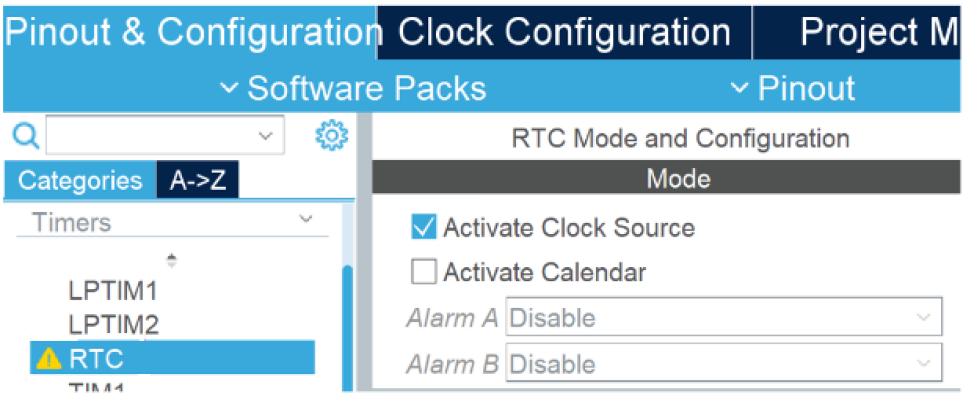


Рисунок 7. Настройка RTC

Все время выполнении программы приложение ожидает ввода пользователя с помощью сенсерного резистивного экрана.

Далее, после нажатие на любую из клавиш, находящихся на экране, программа вызывает обработчик нажатий, который определяет дальнейшее поведение.

В зависимости от нажатой клавиши, МК может выполнить следующие действия (см. рис. 14):

- Вычислить выражение, находящееся на экране калькулятора, в случае если пользователь нажимает на кнопку «=» (в любом случае) или на одну из клавиш «+», «-», «*», «/», когда калькулятор находится в режиме «Серия операций».

  Далее МК выводит полученный результат на экран;
- Записать, прочитать или изменить значение из памяти или вывести его на экран в качестве основного (действующего) в случае нажатия клавиш «MRC», «M+», «M-»;
- Просто вывести введенный символ на экран в случае нажатие на [0; 9] или «.»;

Далее, МК возвращается в изначальное состояние ожидания ввода пользователя.

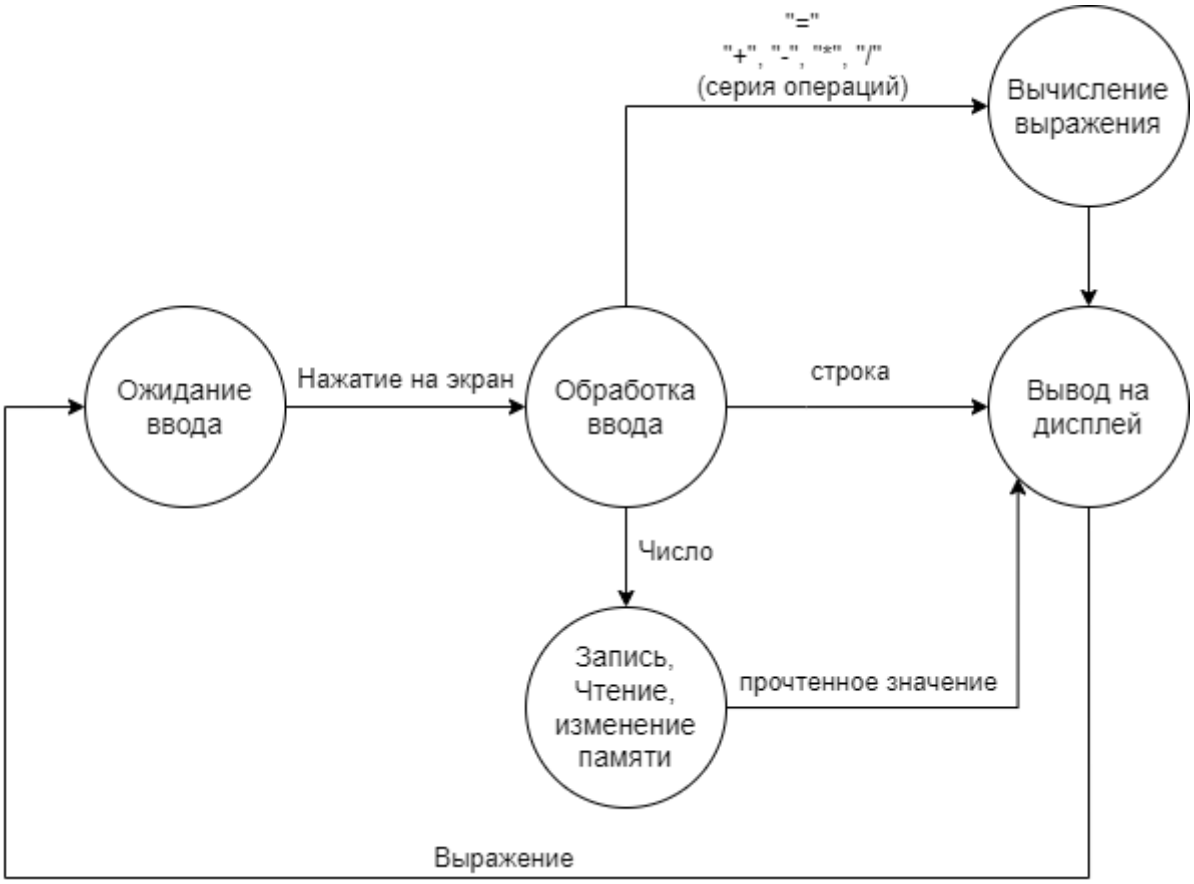


Рисунок 8. Диаграмма состояний

После сброса система так же может находится в двух состояниях:

- В памяти есть сохраненное число;
- В памяти ничего нет;

В зависимости от состояния система будет выводить на экран пометку («M» в углу экрана) об этом и при следующем чтение вернет значение соответствующие текущему состоянию.

Рассмотрим работу полной системы на обобщенной диаграмме потока управления программы

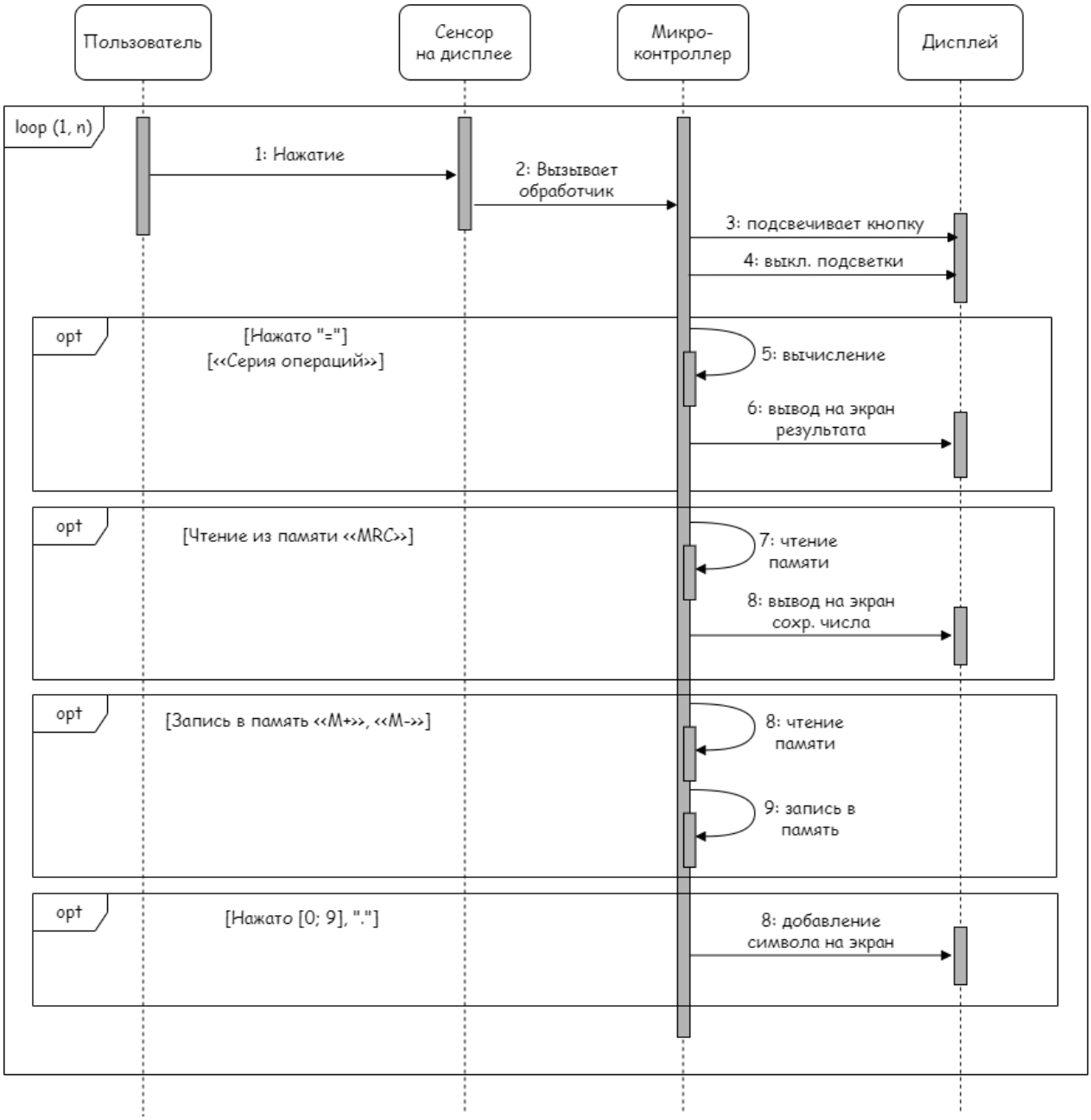


Рисунок 9. Диаграмма последовательности взаимодействия

1) Для передачи изображения от библиотеки на экран необходимо будем использовать линии CS, RS, WR, RD.

- CS – линия выбора чипа. Состояния по умолчанию - логическая 1. Каждый раз, когда происходит взаимодействие с устройством, ее нужно переводить в логический 0.
- RS – выбор режима передачи данных/адреса регистра. Это необходимо, так как интерфейс i80 использует одну шину для передачи адреса регистра и значений самих регистров. Если RS установлена в

логическую 1 – шина используется для передачи данных, в 0 – для передачи адреса регистра.

- WR – линия сигнала записи данных. Состояния по умолчанию – 1. Когда MCU выставил данные на шину, необходимо подать короткий, нулевой импульс, чтобы ILI9325 считал их.
- RD – линия сигнала чтения данных. Состояния по умолчанию – 1. Работает по аналогии с WR, но используется, когда нужно получить данные от ILI9325.

2) Для Сенсор дисплея управляется чипом XPT2046. Для взаимодействия с ним используется SPI интерфейс. Для подключения он требует 4 провода:

1. MOSI — выход ведущего, вход ведомого (Master Out Slave In). Служит для передачи данных от ведущего устройства ведомому.
2. MISO — вход ведущего, выход ведомого (Master In Slave Out). Служит для передачи данных от ведомого устройства ведущему.
3. SCLK или SCK — последовательный тактовый сигнал (Serial Clock). Служит для передачи тактового сигнала для ведомых устройств.
4. CS или SS — выбор микросхемы, выбор ведомого (Chip Select, Slave Select).

С помощью SPI можно подключить одно или несколько устройств, выделив для каждого отдельный CS пин (рисунок 10).

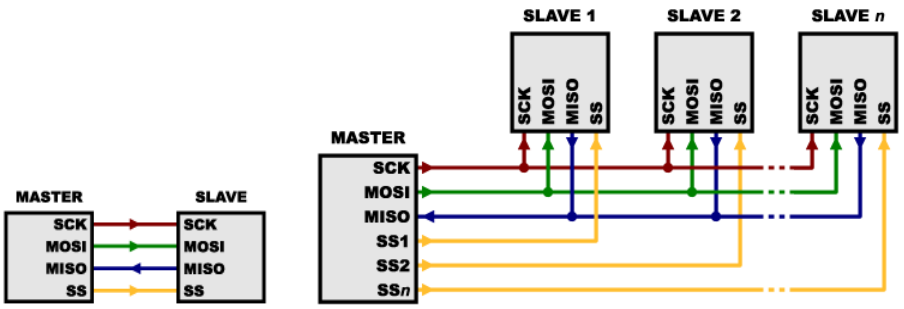


Рисунок 10. Подключение одного или нескольких SPI устройств

Типичная схема передачи данных по SPI представлена на рисунке 11. Из особенностей SPI отметим следующие:

- он позволяет одновременно передавать и получать данные;
- размер пакета составляет 1 байт (8 бит).
- интерфейс широко распространен и реализован в множестве микроконтроллеров, в том числе в STM32 с учетом требований к таймингам тактового сигнала и сигналам с данными.

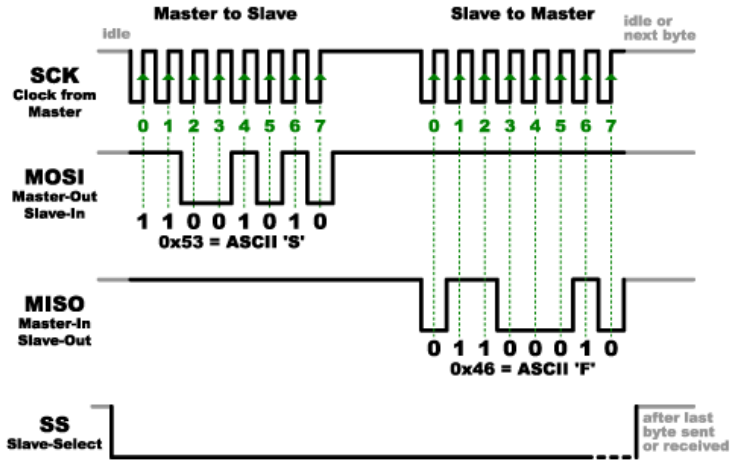


Рисунок 11. Передача данных по SPI

## 4. Описание алгоритмов

Детально разберем функцию передачи изображения от библиотеки на экран с помощью блок-схемы:

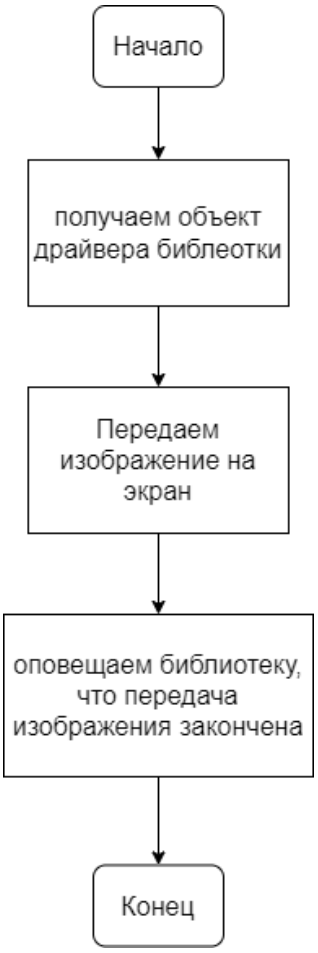


Рисунок 12. Алгоритм передачи изображения

В данном случае, наша программа выступает в качестве посредника между аппаратным обеспечением и функциями библиотеки LVGL.

Также, рассмотрим и функцию считывания текущего положения нажатия на сенсорный резистивный экран.

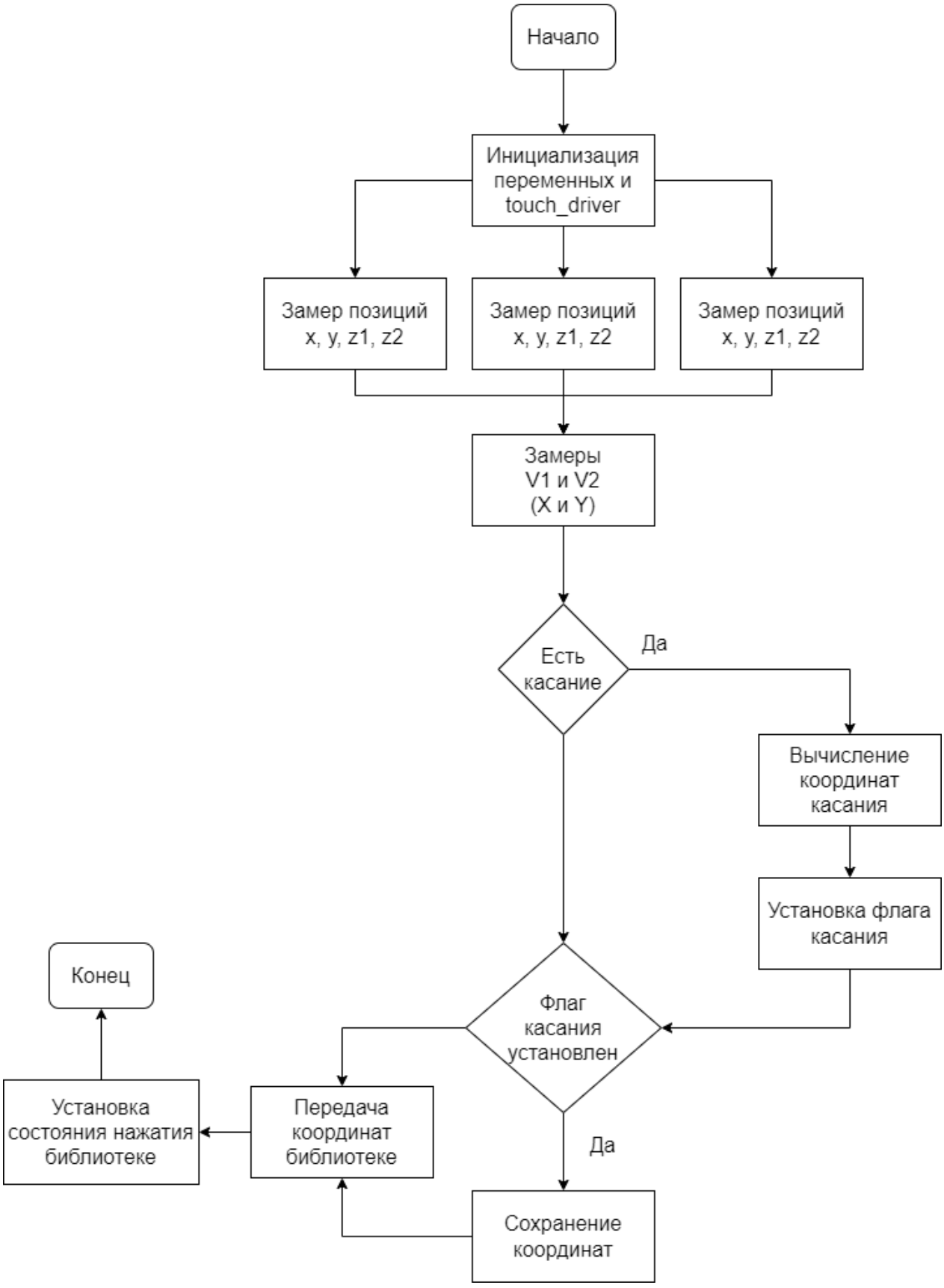


Рисунок 13. Алгоритм считывания нажатия

Детализируем состояния системы на блок-схеме основной функции-обработчика `keyboard_event_handler`, отвечающий за все нажатия по виртуальной клавиатуре:

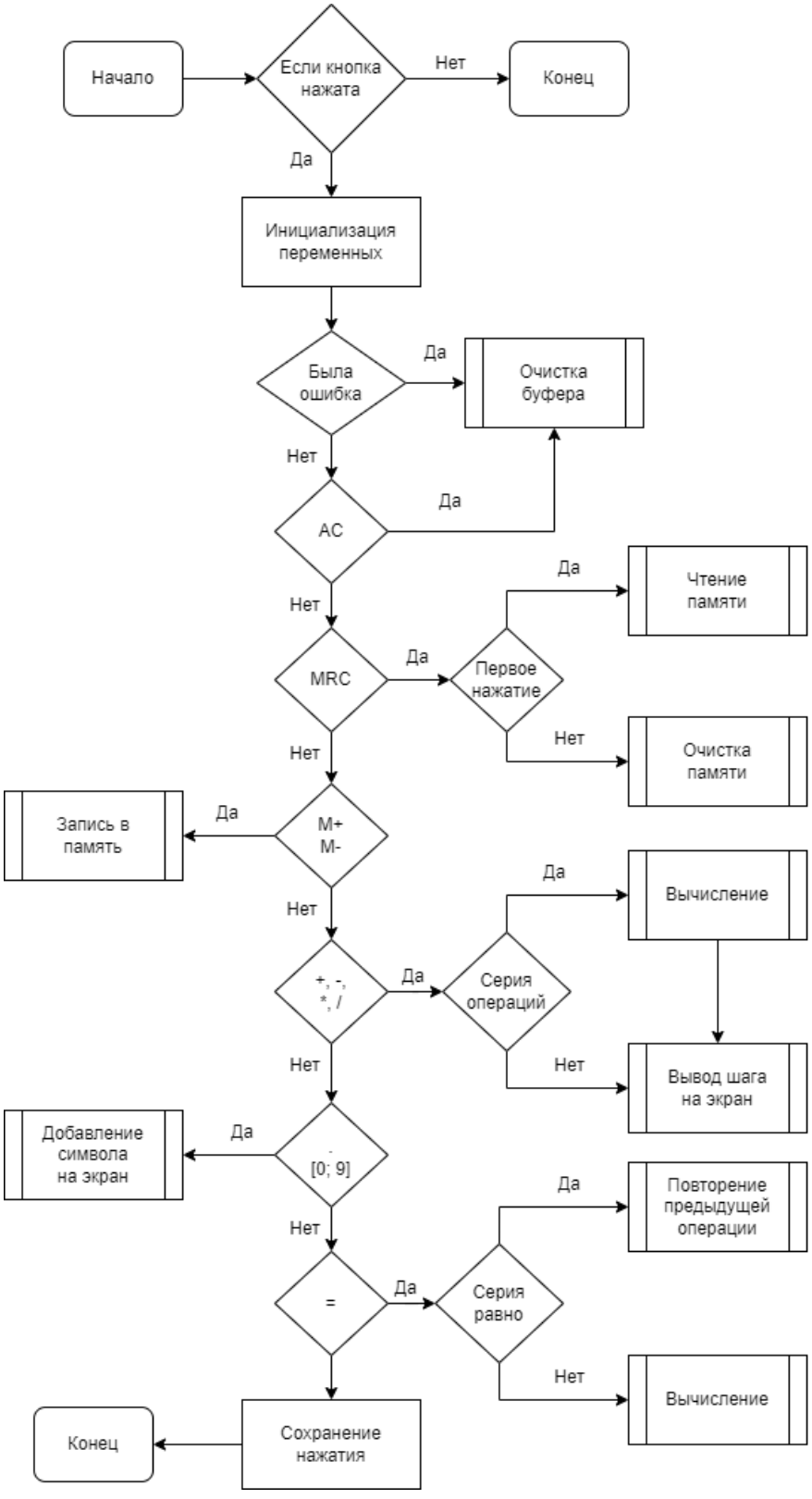


Рисунок 14. Алгоритм обработки клавиш калькулятора

Отсюда следует выделить следующие основные управляющие структуры:

1) Функция `keyboard_event_handler` вызывается в бесконечном цикле вызова `lv_task_handler();`
2) Функция `all_clear` (`«Очистка буфера»`) очищает все поля на экране, выделенную память под переменные и устанавливает исходное состояние программы, кроме статуса памяти;
3) Функция `read_mem` (`«Чтение памяти»`) считывает данные с указанного RTC регистра резервного копирования (опционально с FLASH памятью);
4) Функция `write_mem` (`«Запись в память»`) записывает данные в указанный RTC регистр резервного копирования (опционально с FLASH памятью);
5) Функция `eval` (`«Вычисление»`) принимает два операнда и оператор, который необходимо применить к ним. Далее, совершает необходимые проверки (если они нужны) и выполняет арифметическую операцию, введенную пользователем;
6) Функция `save_display` (`«Вывод шага на экран»`) принимает оператор и текущий текст (число) на экране и форматирует их, перенося в верхнее (дополнительное) текстовое поле;
7) Функция (`«Повторение предыдущей операции»`) вызывается в случае режима «Серия равно» и совершает еще раз последнюю операцию над новым операндом (результат последней операции) и старым операндом;

Рассмотрим подробнее алгоритм чтения и записи в память на примере блок-схемы:

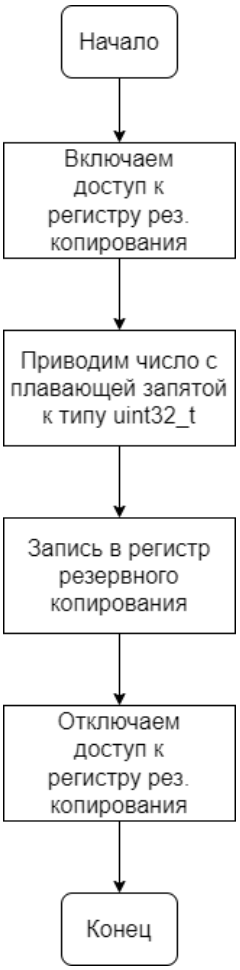


Рисунок 15. Запись в память


Рисунок 16. Чтение из памяти

В программе «Калькулятор» предусмотрено три основным «режима» работы:

1) Обычный – выполнений простых операций без дополнительных условий.
   Пример нажатий: 2, +, 2, = (На экране 4);
2) Серия равно – при неоднократном нажатии клавиши «=» программа повторяет последнюю арифметическую операцию, но над последним результатом операции.
   Пример: 2, +, 2, = (На экране 4), = (На экране 6), = (На экране 8) и так далее;
3) Серия операций – при неоднократном выполнении арифметической операции, игнорируя клавишу «=».
   Пример: 4, +, 2, + (На экране 6 +), 2, + (На экране 8) и так далее;

За режимы работы и аналогичные исключительные ситуации в программе отвечает флаговая переменная и флаговое перечисление:

- SQ_FUN – для метки серии операций;
- SQ_EQ – для метки серии равно;
- MEM_SAVE – для метки наличия сохраненной информации в памяти;
- ERR – отметка об ошибке (в случае ошибки выводится сообщение и далее очищается буфер);

# 5. Тестирование

Тесты для контроля соответствия прибора техническому заданию:

## 5.1 Сложение

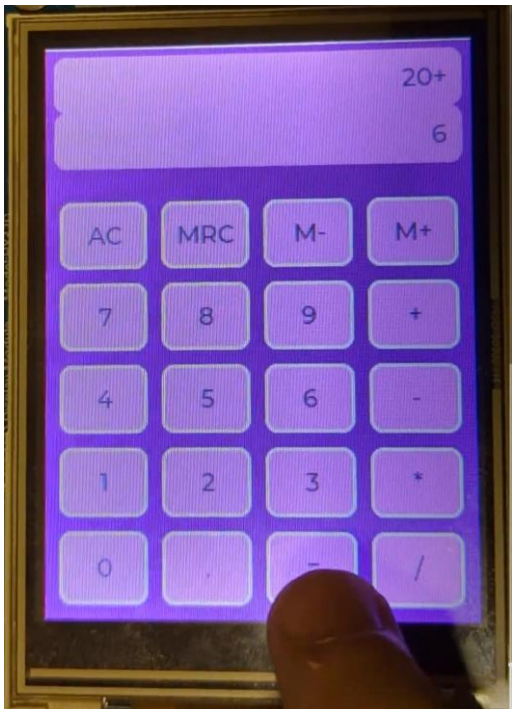


Рисунок 17. До сложения


Рисунок 18. После сложения

## 5.2 Вычитание

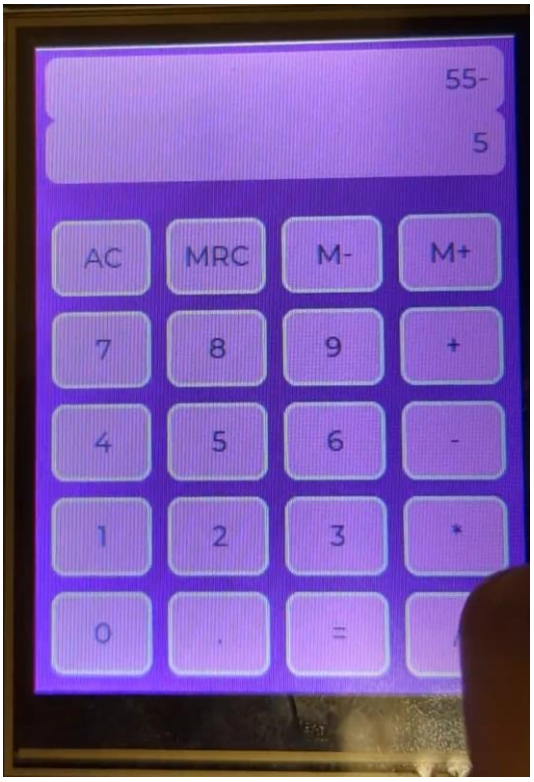


Рисунок 19. До вычитания

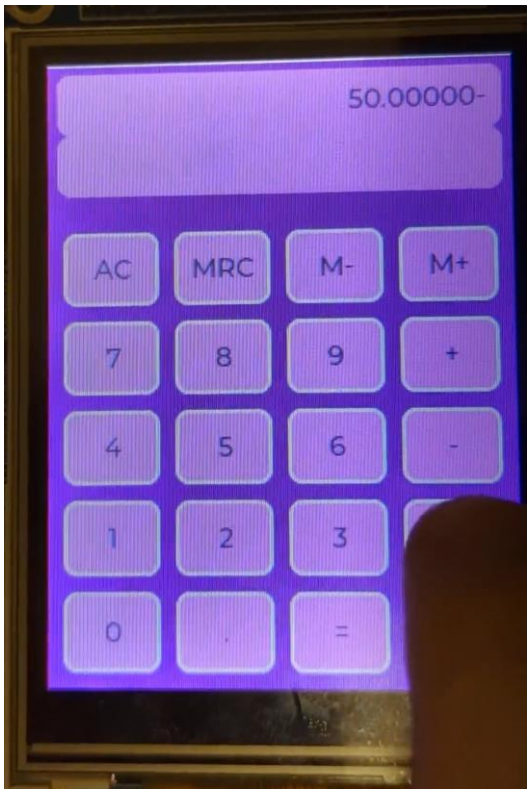


Рисунок 20 После вычитания

## 5.3 Умножение

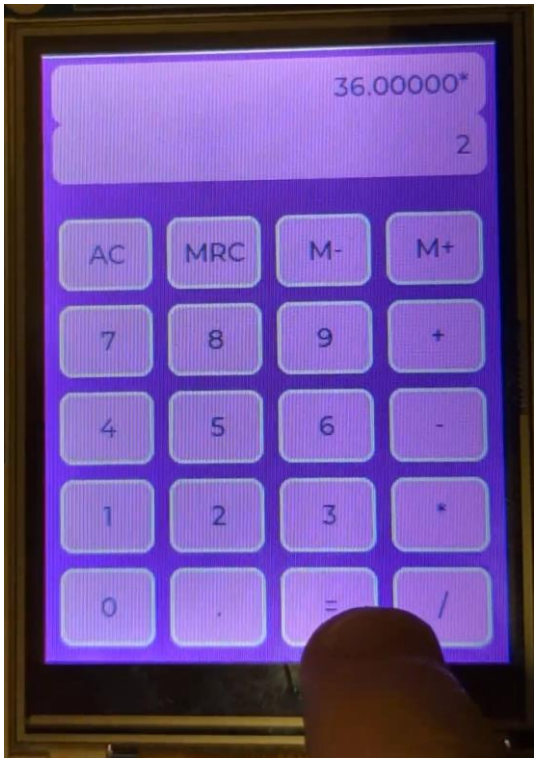


Рисунок 21. До умножения

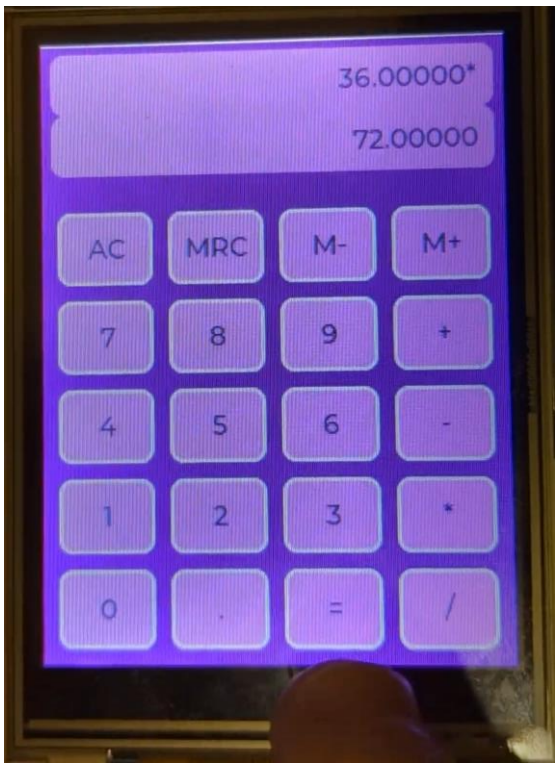


Рисунок 22. После умножения

## 5.4 Деление

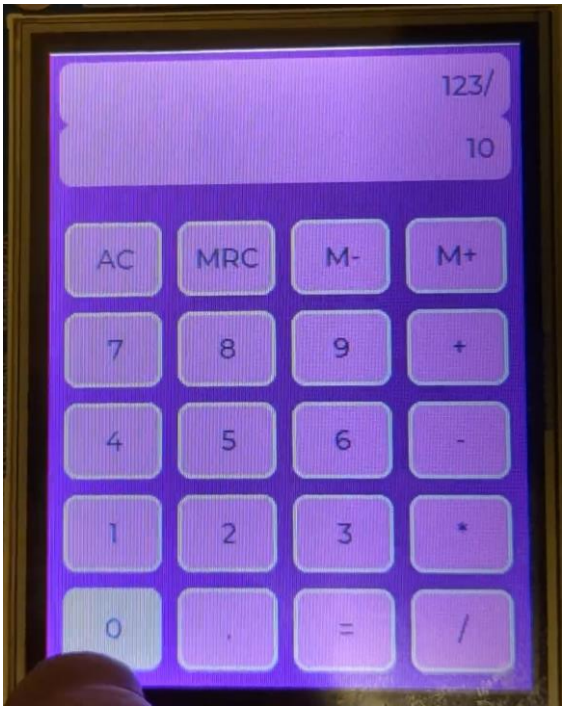


Рисунок 23. До деления


Рисунок 24. После деления

## 5.5 Запись в память


Рисунок 25. До записи


Рисунок 26. После записи

## 5.6 Чтение из памяти


Рисунок 27. До чтения

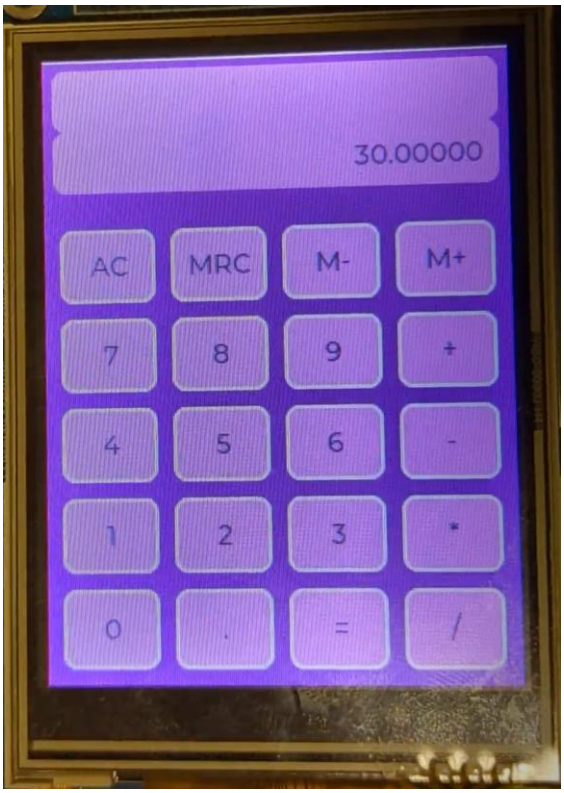


Рисунок 28. После чтения

## 5.7 Очистка

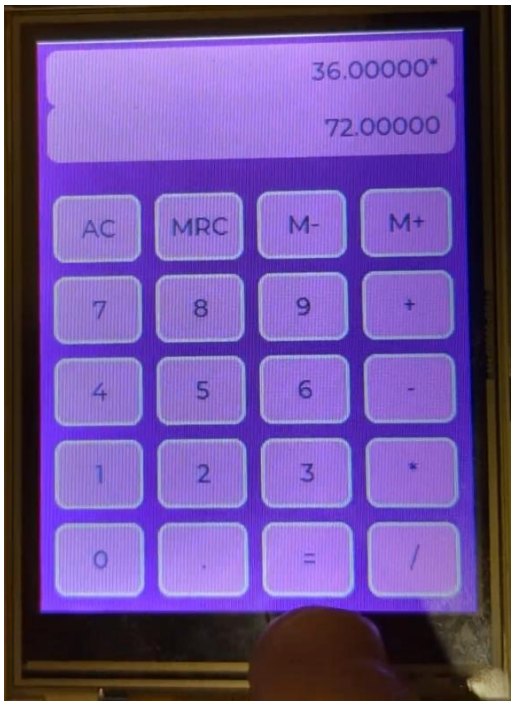


Рисунок 29. До очистки


Рисунок 30. После очистки

### 5.8 Итог

В ходе тестирования программы не было выявлено критических ошибок, влияющих на корректную работу приложения.

Однако, при более комплексном как позитивном, так и негативном тестировании, есть вероятность обнаружить незначительную ошибку / недочет, который в определенной ситуации может вызвать неверный результат или поведения от программы.

## 6. Экономическая оценка

| Компонент | Цена на Aliexpress | Ссылка на Aliexpress |
|---|---|---|
| Микроконтроллер | 82 руб. | Отладочная плата STM32F103C6 |
| Сенсорный экран | 176 руб. | TFT ЖК-экран сенсорный экран ILI9325 |
| Итог | 258 руб | Без учета сборки, проводов и корпуса. |

При массовом производстве, цена на компоненты будет ниже из-за оптовых покупок, что является одним из путей снижения цены.

## 7. Заключение

В результате выполнения данного проекта было осуществлено проектирование приложения на базе микроконтроллера STM32F303VCT6. Были получены основные навыки в области архитектуры и программного обеспечения встроенных систем, а также знания о структуре, функциях и основах программирования микроконтроллеров, позволяющих решать вопросы анализа функционирования программного обеспечения встраиваемых систем. Также, подробно разобраны принципы работы библиотеки LVGL версии 7.11 и ее взаимодействия с полноценной программой.

## 8. Список используемой литературы


1. Микроконтроллеры. Разработка встраиваемых приложений: учебное пособие / А.Е. Васильев; С.-Петербургский государственный политехнический ун-т. - СПб. : Изд-во СПбГПУ, 2003. - 211 с.

2. Лабораторный практикум по изучению ARM микроконтроллеров серии STM32: учебное пособие / А.А. Ключарев, К.А. Кочин; С.-Петербургский государственный ун-т. Аэрокосмического приборостроения - СПб. : Изд-во СПбГУАП, 2022.

3. The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors. Third Edition. Joseph Yiu. ARM Ltd., Cambridge, UK. [электронный ресурс] // URL: https://www.pdfdrive.com/the-definitive-guide-to-arm-cortex-m3-and-cortex-m4processors-e187111520.html

4. Джозеф Ю. Ядро Cortex-M3 компании ARM. Полное руководство. 2012. ISBN: 978- 5-94120-243-0. [электронный ресурс] // URL: https://bok.xyz/book/2373589/b5c3ad

5. RM0008. Reference manual STM32F101xx, STM32F102xx, STM32F103xx, STM32F105xx and STM32F107xx advanced Arm®-based 32-bit MCUs [электронный ресурс] // URL: https://www.st.com/resource/en/reference_manual/cd00171190-stm32f101xxstm32f102xx-stm32f103xx-stm32f105xx-and-stm32f107xx-advanced-armbased32bit-mcus-stmicroelectronics.pdf

6. Datasheet STM32F103x8 STM32F103xB Medium-density performance line ARM®- based 32-bit MCU with 64 or 128 KB Flash, USB, CAN, 7 timers, 2 ADCs, 9 com. Interfaces. [электронный ресурс] // URL: https://www.st.com/resource/en/datasheet/stm32f103c8.pdf

7. Мартин М. Инсайдерское руководство по STM32 [электронный ресурс] // URL: https://istarik.ru/file/STM32.pdf

8. LVGL Documentation v7.11.0. [электронный ресурс] // URL: https://docs.lvgl.io/7.11/

# Приложение А. Код программы

Для удобства реализация проекта находится в открытом репозитории на GitHub

## Функция передачи изображения библиотеки

```
/**
 * Колбек для передачи отрисованного интерфейса на дисплей.
 *
 * LVGL сам вызовет его, когда ему понадобиться передать данные на дисплей.
 */
static void app_open32f3_lvgl_display_flush_cb(lv_disp_drv_t *disp_drv,
                const lv_area_t *area, lv_color_t *color_p) {
        // получить app_open32f3_lvgl_driver_t объект
        app_open32f3_lvgl_driver_t *lvgl_data = disp_drv->user_data;
        // передать изображение на экран
        lcd_ili93xx_fill_area(lvgl_data->lcd_driver, area->x1, area->y1, area->x2,
                        area->y2, (uint16_t*) color_p);
        // оповестить LVGL, что передача изображения на экран закончена
        lv_disp_flush_ready(disp_drv);
}
```

## Функция обработки касания

```
/**
 * Колбек для получения текущих координат касания от сенсорного дисплей.
 *
 * LVGL сам вызовет его, когда ему понадобиться считать данные ввода.
 */
static bool app_open32f3_lvgl_display_touch_sensor_read_cb(
                lv_indev_drv_t *disp_drv, lv_indev_data_t *data) {
        // получить app_open32f3_lvgl_driver_t объект
        app_open32f3_lvgl_driver_t *lvgl_data = disp_drv->user_data;

        int touch_x = 0;
        int touch_y = 0;
        int touch_flag = 0;
        lcd_xpt2046_driver_t *touch_driver = lvgl_data->touch_driver;

        int x1, y1, z11, z21, x2, y2, z12, z22, x3, y3, z13, z23,
                Z1disp = 0, Z2disp = 0;

        lcd_xpt2046_measure(touch_driver, XPT2046_CMD_MEASURE_X, &x1);
        lcd_xpt2046_measure(touch_driver, XPT2046_CMD_MEASURE_Y, &y1);
        lcd_xpt2046_measure(touch_driver, XPT2046_CMD_MEASURE_Z1, &z11);
        lcd_xpt2046_measure(touch_driver, XPT2046_CMD_MEASURE_Z2, &z21);

        lcd_xpt2046_measure(touch_driver, XPT2046_CMD_MEASURE_X, &x2);
        lcd_xpt2046_measure(touch_driver, XPT2046_CMD_MEASURE_Y, &y2);
        lcd_xpt2046_measure(touch_driver, XPT2046_CMD_MEASURE_Z1, &z12);
        lcd_xpt2046_measure(touch_driver, XPT2046_CMD_MEASURE_Z2, &z22);

        lcd_xpt2046_measure(touch_driver, XPT2046_CMD_MEASURE_X, &x3);
        lcd_xpt2046_measure(touch_driver, XPT2046_CMD_MEASURE_Y, &y3);
        lcd_xpt2046_measure(touch_driver, XPT2046_CMD_MEASURE_Z1, &z13);
        lcd_xpt2046_measure(touch_driver, XPT2046_CMD_MEASURE_Z2, &z23);

        Z1disp = calculate_touch_position(z11, z12, z13);
        Z2disp = calculate_touch_position(z21, z22, z23);
```

```
        if (abs(Z1disp - Z2disp) < 3800) {                              //ЕСТЬ

                touch_x = calculate_touch_position(x1, x2, x3);
                touch_y = calculate_touch_position(y1, y2, y3);
                touch_x = 0.062 * touch_x ;
                touch_y = 0.084 * touch_y ;
                if (touch_x <= 0) {
                        touch_x = 0;
                }
                if (touch_x >= 240 ) {
                        touch_x = 240 ;
                }
                if (touch_y <= 0) {
                        touch_y = 0;
                }
                if (touch_y >= 320 ) {
                        touch_y = 320 ;
                }
                touch_flag = 1;
        }

        // !!! ================================================================
        // Определить, имеется ли касание и записать результат во флаг touch_flag
        // Если касание произошло, то записать его координаты в touch_x и touch_y
        // !!! ================================================================

        // обработка логики устройства ввода в соответствии с LVGL
        if (touch_flag) {
                lvgl_data->last_x_position = touch_x;
                lvgl_data->last_y_position = touch_y;
        }
        data->point.x = lvgl_data->last_x_position;
        data->point.y = lvgl_data->last_y_position;
        if (touch_flag) {
                // касание дисплея обнаружено
                data->state = LV_INDEV_STATE_PR;
        } else {
                // касание дисплея не обнаружено
                data->state = LV_INDEV_STATE_REL;
        }
        return false;
}
```

# Функция обработки нажатия на клавиатуру

```
static void keyboard_event_handler(lv_obj_t * obj, lv_event_t event)
{
    if(event == LV_EVENT_VALUE_CHANGED) {
        uint16_t id = lv_btnmatrix_get_active_btn(obj);
        static uint16_t prev_id = 0;
        static char* first_arg;
        // 0 bit - sequence of functions
        // 1 bit - sequence of equality
        // 2 bit - memory first save
        // 3 bit - MRC toggle
        // 4 bit - error state
        static e_function fun = NONE;
        const char* txt = lv_btnmatrix_get_active_btn_text(obj);
        const char* text_on_disp = strdup(lv_textarea_get_text(app_scene.text_main));
        /*
         *      AC_0    MRC_1   M-_2    M+_3
         *      7_4             8_5             9_6             +_7
         *      4_8             5_9             6_10    -_11
         *      1_12    2_13    3_14    *_15
         *      0_16    ._17    =_18    /_19
```

```
         *
         */
        if (get_flag(ERR))
        {
        set_flag(ERR, false);
                        all_clear(&fun, &first_arg, &text_on_disp);
        }

        switch (id)
        {
        case 0: // AC
                all_clear(&fun, &first_arg, &text_on_disp);
                break;
        case 1: // MRC
                if (get_flag(MEM_SAVE) && prev_id == id)
                {
                        set_flag(MEM_SAVE, false);
                        write_mem(.0f);
                        lv_label_set_text(app_scene.text_mem, "");
                }
                else if (get_flag(MEM_SAVE))
                {
                        //mem_value = read_mem();
                        //format_result(text_on_disp, mem_value);
                        format_result(text_on_disp, read_mem());
                        lv_textarea_set_text(app_scene.text_main, text_on_disp);
                }
                break;
        case 2: // M-
                memory_operation(text_on_disp, MINUS);
                break;
        case 3: // M+
                memory_operation(text_on_disp, PLUS);
                break;

        case 7: // +
                fun_execute(&first_arg, text_on_disp, &fun, PLUS);
                break;
        case 11: // -
                fun_execute(&first_arg, text_on_disp, &fun, MINUS);
                break;
        case 15: // *
                fun_execute(&first_arg, text_on_disp, &fun, MUL);
                break;
        case 19: // /
                fun_execute(&first_arg, text_on_disp, &fun, DIV);
                break;

        case 17: // .
                if (strlen(text_on_disp) == 0)
                        lv_textarea_set_text(app_scene.text_main, "0.");
                else if (!is_char_contain('.', text_on_disp))
                        lv_textarea_add_text(app_scene.text_main, txt);
            break;

        case 18: // =
                if (get_flag(SQ_FUN))
                {
                        lv_textarea_set_text(app_scene.text_main, eval(first_arg, text_on_disp, fun));
                        if (fun != MUL)
                                                first_arg = strdup(text_on_disp);
                        save_display(first_arg, fun);
                        set_flag(SQ_FUN, false);
                        set_flag(SQ_EQ, true);
                }
                else if (get_flag(SQ_EQ))
                                {
                        if (fun == MUL)
                                lv_textarea_set_text(app_scene.text_main, eval(first_arg, text_on_disp,
fun));
                        else
                                lv_textarea_set_text(app_scene.text_main, eval(text_on_disp, first_arg,
fun));
                                }
                break;

        case 4:
```

```
        case 5:
        case 6:
        case 8:
        case 9:
        case 10:
        case 12:
        case 13:
        case 14:
        case 16:
                lv_textarea_add_text(app_scene.text_main, txt);
                break;
        }
        prev_id = id;
    }
}
```

## Функция очистки

```
void all_clear(e_function* fun, char **first_arg, char** text_main) {

        free(*first_arg);

        free(*text_main);

        lv_textarea_set_text(app_scene.text_main, "");

        lv_textarea_set_text(app_scene.text_history, "");

        (*fun) = NONE;

}
```

## Функции вычисления результата

```
const char* eval(const char* left, const char* right, e_function fun)

{

        float left_f = atof(left);

        float right_f = atof(right);

        char buf[MAX_LENGTH];

        switch(fun)

        {

                case PLUS:

                        format_result(buf, left_f + right_f);

                        break;

                case MINUS:

                        format_result(buf, left_f - right_f);

                        break;

                case MUL:

                        format_result(buf, left_f * right_f);

                        break;

                case DIV:

                        if (float_equal(0.f, right_f, 0.00000001f))

                        {

                                set_flag(ERR, true);

                                strcpy(buf, "ERROR: Division by 0");

                        }

                        else

                                format_result(buf, left_f / right_f);
```

```
                        break;
                case NONE:
                        strcpy(buf, right);
                        break;
        }


        return strdup(buf);
}
```

## Функции сохранения значения в память

```
void write_mem(float num)
{
    HAL_PWR_EnableBkUpAccess();

    uint32_t fbits = 0;
    memcpy(&fbits, &num, sizeof fbits);
    HAL_RTCEx_BKUPWrite(&hrtc, RTC_BKP_DR0, fbits);

    HAL_PWR_DisableBkUpAccess();
}
```

## Функции чтения значения из памяти

```
float read_mem()
{
        uint32_t value = HAL_RTCEx_BKUPRead(&hrtc, RTC_BKP_DR0);
        float fvalue = .0f;
        memcpy(&fvalue, &value, sizeof fvalue);
        return fvalue;
}
```