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

Дисциплина: Проектирование вычислительных систем

Лабораторная работа №5

Вариант 4

Выполнили:
Барсуков Максим Андреевич,
группа P3415
Стригалев Никита Сергеевич,
группа P3412

Преподаватель:
Пинкевич Василий Юрьевич

2025 г.

Санкт-Петербург

Содержание

Задание

Разработать программу, которая включает драйвер OLED-дисплея. Для организации обмена данными с дисплеем можно использовать те же три способа, что и в предыдущей работе. Поскольку дисплей управляется по той же шине I²C, что и клавиатура, необходимо избежать конфликтов доступа к шине. При работе с I²C по опросу они не возникнут, а при работе по прерыванию следует планировать вычисления так, чтобы транзакции не пересекались по времени.

Если транзакции I²C генерируются по прерыванию от таймера, следует встроить транзакции обновления видеопамяти дисплея в расписание опроса клавиатуры. При этом следует учесть, что отправка буфера данных в контроллер дисплея происходит значительно дольше, чем любая транзакция опроса клавиатуры. Например, после восьми вызовов обработчика прерываний для опроса клавиатуры можно инициировать отправку буфера и несколько вызовов обработчика отправлять никаких транзакций по I²C, давая время на завершение отправки буфера. Начальную инициализацию дисплея, как и клавиатуры, удобнее делать в режиме опроса, пока не включены прерывания от таймера.

Вариант: 4

Адаптировать программу-калькулятор для использования с дисплеем и клавиатурой SDK-1.1M. Кнопки клавиатуры использовать для:

- ввода цифр 0 – 9;
- ввода операции: тип операции выбирается нажатием отдельной кнопки (одно нажатие – «+», два – «-», три – «*», четыре – «/», далее снова происходит переход к «+»), текущая выбранная операция запоминается при начале ввода второго операнда;
- окончания ввода и расчета ответа (эквивалент «=»).

На экране отображается три строки:

- первый операнд и знак операции;
- второй операнд и знак «=»;
- результат.

Пример:

```
xxxxx+
yyyyy=
zzzzz
```

При начале ввода первого операнда экран очищается от предыдущих данных.

Используемые контакты

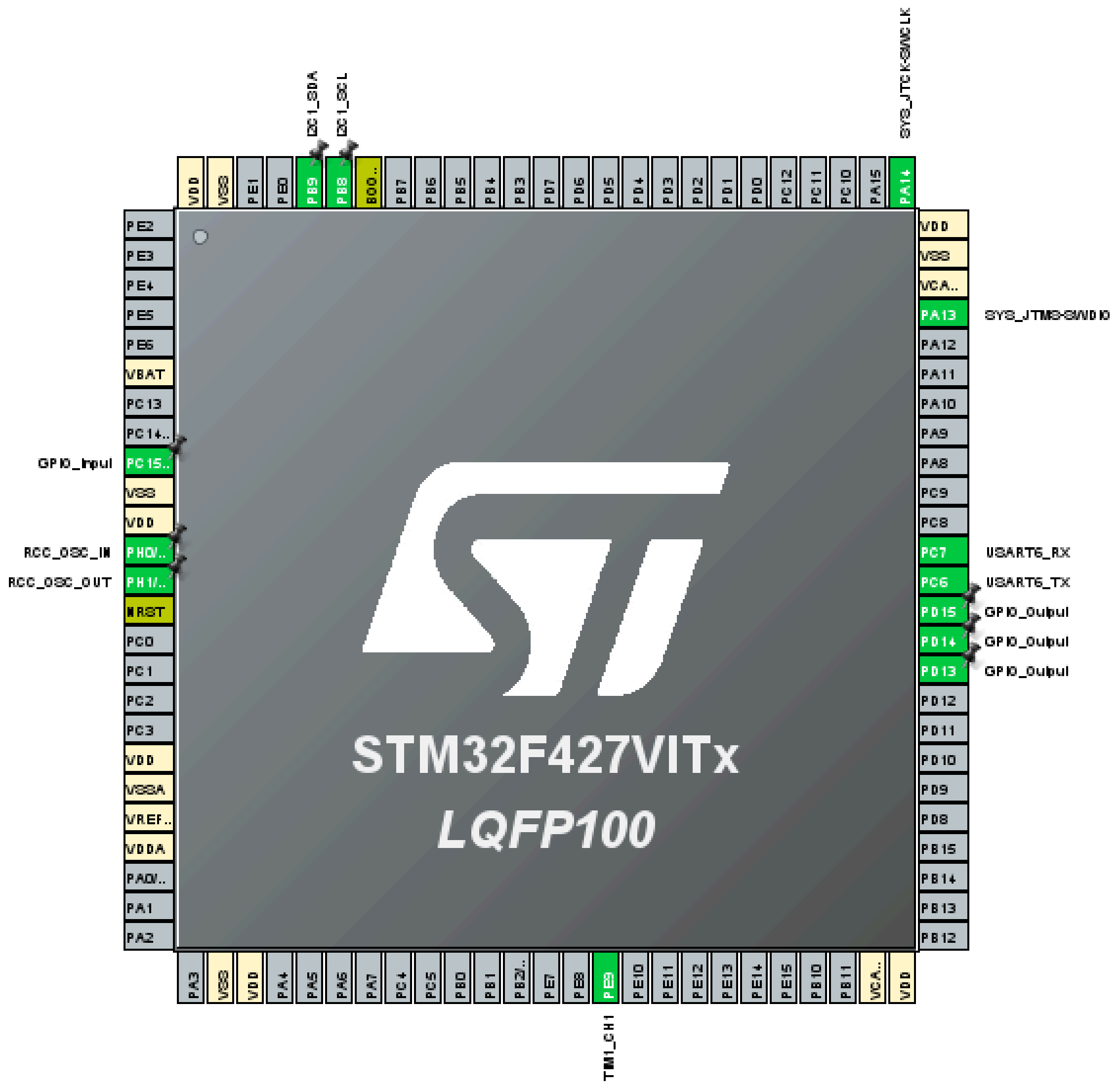

Рисунок 1 – Используемые контакты

Блок-схема

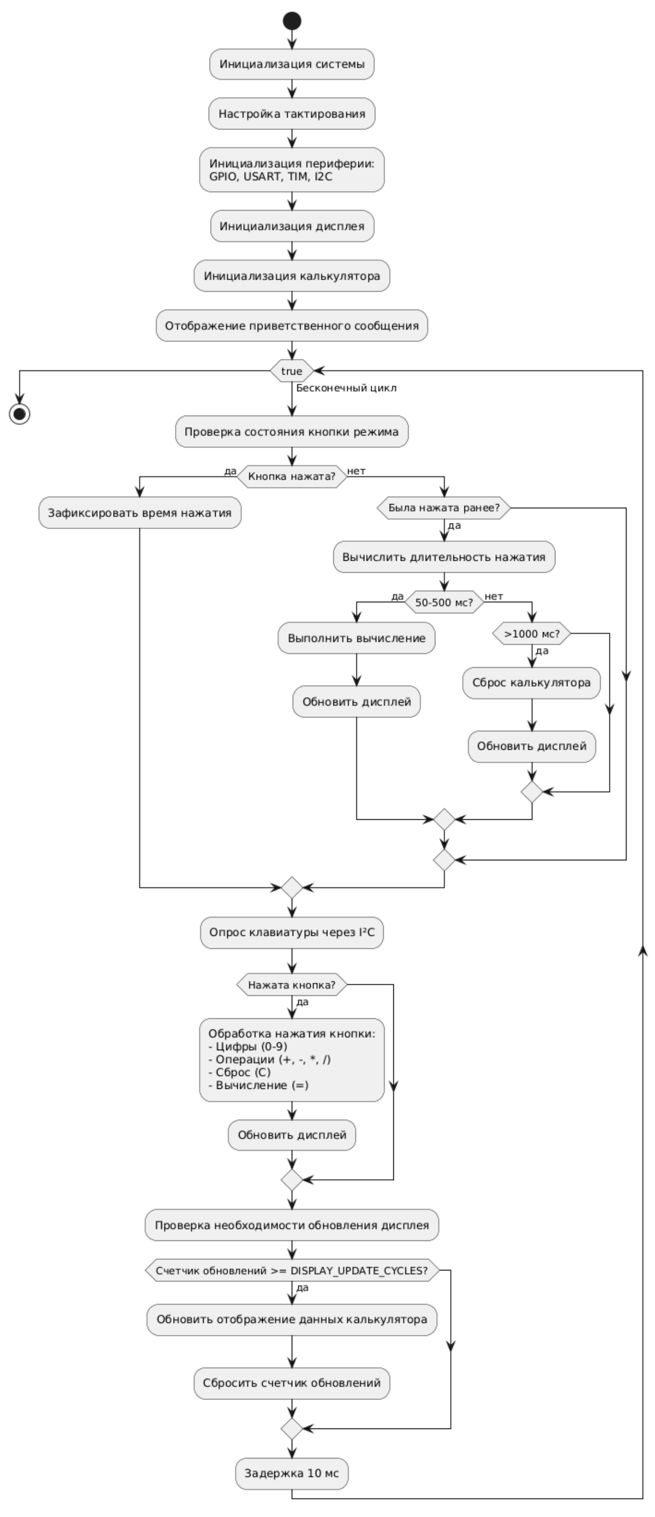
Инициализация системы
Настройка тактирования
Инициализация периферии: GPIO, USART, TIM, I2C
Инициализация дисплея
Инициализация калькулятора
Отображение приветственного сообщения
true
Бесконечный цикл
Проверка состояния кнопки режима
да
Кнопка нажата?
нет
Зафиксировать время нажатия
Была нажата ранее?
да
Вычислить длительность нажатия
да
50-500 мс?
нет
Выполнить вычисление
>1000 мс?
да
Сброс калькулятора
Обновить дисплей
Обновить дисплей
Опрос клавиатуры через I²C
Нажата кнопка?
да
Обработка нажатия кнопки:
- Цифры (0-9)
- Операции (+, -, *, /)
- Сброс (С)
- Вычисление (=)
Обновить дисплей
Проверка необходимости обновления дисплея
Счетчик обновлений >= DISPLAY_UPDATE_CYCLES?
да
Обновить отображение данных калькулятора
Сбросить счетчик обновлений
Задержка 10 мс

Рисунок 2 – Схема программы

Описание функций кнопок

Символ	Действие
«0» – «9»	Ввод цифры операнда.
«e»	Меняет операцию: «+» ⟶ «–» ⟶ «*» ⟶ «/» и так по кругу.
«x»	Очищает калькулятор, ставит операнды в 0, операцию в «None».

Раскладка

1	2	3
4	5	6
7	8	9
0	e	x

Исходный код

display.c

```
#include "display.h"
#include "i2c.h"

static uint8_t display_buffer[DISPLAY_WIDTH * DISPLAY_HEIGHT / 8];

static DisplayState display_state;

static I2C_HandleTypeDef *i2c_handle = NULL;

static const uint16_t font_small_data[] = {
          0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000,  // sp
          0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x0000, 0x1000,
0x0000, 0x0000,  // !
          0x2800, 0x2800, 0x2800, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000,  // "
          0x2400, 0x2400, 0x7C00, 0x2400, 0x4800, 0x7C00, 0x4800, 0x4800,
0x0000, 0x0000,  // #
          0x3800, 0x5400, 0x5000, 0x3800, 0x1400, 0x5400, 0x5400, 0x3800,
0x1000, 0x0000,  // $
          0x2000, 0x5400, 0x5800, 0x3000, 0x2800, 0x5400, 0x1400, 0x0800,
0x0000, 0x0000,  // %
          0x1000, 0x2800, 0x2800, 0x1000, 0x3400, 0x4800, 0x4800, 0x3400,
0x0000, 0x0000,  // &
          0x1000, 0x1000, 0x1000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000,  // '
          0x0800, 0x1000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000, 0x2000,
0x1000, 0x0800,  // (
          0x2000, 0x1000, 0x0800, 0x0800, 0x0800, 0x0800, 0x0800, 0x0800,
0x1000, 0x2000,  // )
          0x1000, 0x3800, 0x1000, 0x2800, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000,  // *
          0x0000, 0x0000, 0x1000, 0x1000, 0x7C00, 0x1000, 0x1000, 0x0000,
0x0000, 0x0000,  // +
          0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x1000,
0x1000, 0x1000,  // ,
          0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x3800, 0x0000, 0x0000,
0x0000, 0x0000,  // -
          0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x1000,
0x0000, 0x0000,  // .
          0x0800, 0x0800, 0x1000, 0x1000, 0x1000, 0x1000, 0x2000, 0x2000,
0x0000, 0x0000,  // /
          0x3800, 0x4400, 0x4400, 0x5400, 0x4400, 0x4400, 0x4400, 0x3800,
0x0000, 0x0000,  // 0
          0x1000, 0x3000, 0x5000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000,
0x0000, 0x0000,  // 1
          0x3800, 0x4400, 0x4400, 0x0400, 0x0800, 0x1000, 0x2000, 0x7C00,
0x0000, 0x0000,  // 2
```

```
             0x3800, 0x4400, 0x0400, 0x1800, 0x0400, 0x0400, 0x4400, 0x3800,
0x0000, 0x0000,  // 3
             0x0800, 0x1800, 0x2800, 0x2800, 0x4800, 0x7C00, 0x0800, 0x0800,
0x0000, 0x0000,  // 4
             0x7C00, 0x4000, 0x4000, 0x7800, 0x0400, 0x0400, 0x4400, 0x3800,
0x0000, 0x0000,  // 5
             0x3800, 0x4400, 0x4000, 0x7800, 0x4400, 0x4400, 0x4400, 0x3800,
0x0000, 0x0000,  // 6
             0x7C00, 0x0400, 0x0800, 0x1000, 0x1000, 0x2000, 0x2000, 0x2000,
0x0000, 0x0000,  // 7
             0x3800, 0x4400, 0x4400, 0x3800, 0x4400, 0x4400, 0x4400, 0x3800,
0x0000, 0x0000,  // 8
             0x3800, 0x4400, 0x4400, 0x4400, 0x3C00, 0x0400, 0x4400, 0x3800,
0x0000, 0x0000,  // 9
             0x0000, 0x0000, 0x1000, 0x0000, 0x0000, 0x0000, 0x0000, 0x1000,
0x0000, 0x0000,  // :
             0x0000, 0x0000, 0x0000, 0x1000, 0x0000, 0x0000, 0x0000, 0x1000,
0x1000, 0x1000,  // ;
             0x0000, 0x0000, 0x0C00, 0x3000, 0x4000, 0x3000, 0x0C00, 0x0000,
0x0000, 0x0000,  // <
             0x0000, 0x0000, 0x0000, 0x7C00, 0x0000, 0x7C00, 0x0000, 0x0000,
0x0000, 0x0000,  // =
             0x0000, 0x0000, 0x6000, 0x1800, 0x0400, 0x1800, 0x6000, 0x0000,
0x0000, 0x0000,  // >
             0x3800, 0x4400, 0x0400, 0x0800, 0x1000, 0x1000, 0x0000, 0x1000,
0x0000, 0x0000,  // ?
             0x3800, 0x4400, 0x4C00, 0x5400, 0x5C00, 0x4000, 0x4000, 0x3800,
0x0000, 0x0000,  // @
             0x1000, 0x2800, 0x2800, 0x2800, 0x2800, 0x7C00, 0x4400, 0x4400,
0x0000, 0x0000,  // A
             0x7800, 0x4400, 0x4400, 0x7800, 0x4400, 0x4400, 0x4400, 0x7800,
0x0000, 0x0000,  // B
             0x3800, 0x4400, 0x4000, 0x4000, 0x4000, 0x4000, 0x4400, 0x3800,
0x0000, 0x0000,  // C
             0x7000, 0x4800, 0x4400, 0x4400, 0x4400, 0x4400, 0x4800, 0x7000,
0x0000, 0x0000,  // D
             0x7C00, 0x4000, 0x4000, 0x7C00, 0x4000, 0x4000, 0x4000, 0x7C00,
0x0000, 0x0000,  // E
             0x7C00, 0x4000, 0x4000, 0x7800, 0x4000, 0x4000, 0x4000, 0x4000,
0x0000, 0x0000,  // F
             0x3800, 0x4400, 0x4000, 0x4000, 0x5C00, 0x4400, 0x4400, 0x3800,
0x0000, 0x0000,  // G
             0x4400, 0x4400, 0x4400, 0x7C00, 0x4400, 0x4400, 0x4400, 0x4400,
0x0000, 0x0000,  // H
             0x3800, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x3800,
0x0000, 0x0000,  // I
             0x0400, 0x0400, 0x0400, 0x0400, 0x0400, 0x0400, 0x4400, 0x3800,
0x0000, 0x0000,  // J
             0x4400, 0x4800, 0x5000, 0x6000, 0x5000, 0x4800, 0x4800, 0x4400,
0x0000, 0x0000,  // K
             0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x4000, 0x7C00,
0x0000, 0x0000,  // L
             0x4400, 0x6C00, 0x6C00, 0x5400, 0x4400, 0x4400, 0x4400, 0x4400,
0x0000, 0x0000,  // M
```

```
        0x4400, 0x6400, 0x6400, 0x5400, 0x5400, 0x4C00, 0x4C00, 0x4400,
0x0000, 0x0000,  // N
        0x3800, 0x4400, 0x4400, 0x4400, 0x4400, 0x4400, 0x4400, 0x3800,
0x0000, 0x0000,  // O
        0x7800, 0x4400, 0x4400, 0x4400, 0x7800, 0x4000, 0x4000, 0x4000,
0x0000, 0x0000,  // P
        0x3800, 0x4400, 0x4400, 0x4400, 0x4400, 0x4400, 0x5400, 0x3800,
0x0400, 0x0000,  // Q
        0x7800, 0x4400, 0x4400, 0x4400, 0x7800, 0x4800, 0x4800, 0x4400,
0x0000, 0x0000,  // R
        0x3800, 0x4400, 0x4000, 0x3000, 0x0800, 0x0400, 0x4400, 0x3800,
0x0000, 0x0000,  // S
        0x7C00, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000,
0x0000, 0x0000,  // T
        0x4400, 0x4400, 0x4400, 0x4400, 0x4400, 0x4400, 0x4400, 0x3800,
0x0000, 0x0000,  // U
        0x4400, 0x4400, 0x4400, 0x2800, 0x2800, 0x2800, 0x1000, 0x1000,
0x0000, 0x0000,  // V
        0x4400, 0x4400, 0x5400, 0x5400, 0x5400, 0x6C00, 0x2800, 0x2800,
0x0000, 0x0000,  // W
        0x4400, 0x2800, 0x2800, 0x1000, 0x1000, 0x2800, 0x2800, 0x4400,
0x0000, 0x0000,  // X
        0x4400, 0x4400, 0x2800, 0x2800, 0x1000, 0x1000, 0x1000, 0x1000,
0x0000, 0x0000,  // Y
        0x7C00, 0x0400, 0x0800, 0x1000, 0x1000, 0x2000, 0x4000, 0x7C00,
0x0000, 0x0000,  // Z
        0x1800, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000,
0x1000, 0x1800,  // [
        0x2000, 0x2000, 0x1000, 0x1000, 0x1000, 0x1000, 0x0800, 0x0800,
0x0000, 0x0000,  /* \ */
        0x3000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000,
0x1000, 0x3000,  // ]
        0x1000, 0x2800, 0x2800, 0x4400, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000,  // ^
        0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0xFE00,  // _
        0x2000, 0x1000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000,  // `
        0x0000, 0x0000, 0x3800, 0x4400, 0x3C00, 0x4400, 0x4C00, 0x3400,
0x0000, 0x0000,  // a
        0x4000, 0x4000, 0x5800, 0x6400, 0x4400, 0x4400, 0x6400, 0x5800,
0x0000, 0x0000,  // b
        0x0000, 0x0000, 0x3800, 0x4400, 0x4000, 0x4000, 0x4400, 0x3800,
0x0000, 0x0000,  // c
        0x0400, 0x0400, 0x3400, 0x4C00, 0x4400, 0x4400, 0x4C00, 0x3400,
0x0000, 0x0000,  // d
        0x0000, 0x0000, 0x3800, 0x4400, 0x7C00, 0x4000, 0x4400, 0x3800,
0x0000, 0x0000,  // e
        0x0C00, 0x1000, 0x7C00, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000,
0x0000, 0x0000,  // f
        0x0000, 0x0000, 0x3400, 0x4C00, 0x4400, 0x4400, 0x4C00, 0x3400,
0x0400, 0x7800,  // g
        0x4000, 0x4000, 0x5800, 0x6400, 0x4400, 0x4400, 0x4400, 0x4400,
0x0000, 0x0000,  // h
```

```
            0x1000, 0x0000, 0x7000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000,
0x0000, 0x0000,  // i
            0x1000, 0x0000, 0x7000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000,
0x1000, 0xE000,  // j
            0x4000, 0x4000, 0x4800, 0x5000, 0x6000, 0x5000, 0x4800, 0x4400,
0x0000, 0x0000,  // k
            0x7000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000,
0x0000, 0x0000,  // l
            0x0000, 0x0000, 0x7800, 0x5400, 0x5400, 0x5400, 0x5400, 0x5400,
0x0000, 0x0000,  // m
            0x0000, 0x0000, 0x5800, 0x6400, 0x4400, 0x4400, 0x4400, 0x4400,
0x0000, 0x0000,  // n
            0x0000, 0x0000, 0x3800, 0x4400, 0x4400, 0x4400, 0x4400, 0x3800,
0x0000, 0x0000,  // o
            0x0000, 0x0000, 0x5800, 0x6400, 0x4400, 0x4400, 0x6400, 0x5800,
0x4000, 0x4000,  // p
            0x0000, 0x0000, 0x3400, 0x4C00, 0x4400, 0x4400, 0x4C00, 0x3400,
0x0400, 0x0400,  // q
            0x0000, 0x0000, 0x5800, 0x6400, 0x4000, 0x4000, 0x4000, 0x4000,
0x0000, 0x0000,  // r
            0x0000, 0x0000, 0x3800, 0x4400, 0x3000, 0x0800, 0x4400, 0x3800,
0x0000, 0x0000,  // s
            0x2000, 0x2000, 0x7800, 0x2000, 0x2000, 0x2000, 0x2000, 0x1800,
0x0000, 0x0000,  // t
            0x0000, 0x0000, 0x4400, 0x4400, 0x4400, 0x4400, 0x4C00, 0x3400,
0x0000, 0x0000,  // u
            0x0000, 0x0000, 0x4400, 0x4400, 0x2800, 0x2800, 0x2800, 0x1000,
0x0000, 0x0000,  // v
            0x0000, 0x0000, 0x5400, 0x5400, 0x5400, 0x6C00, 0x2800, 0x2800,
0x0000, 0x0000,  // w
            0x0000, 0x0000, 0x4400, 0x2800, 0x1000, 0x1000, 0x2800, 0x4400,
0x0000, 0x0000,  // x
            0x0000, 0x0000, 0x4400, 0x4400, 0x2800, 0x2800, 0x1000, 0x1000,
0x1000, 0x6000,  // y
            0x0000, 0x0000, 0x7C00, 0x0800, 0x1000, 0x2000, 0x4000, 0x7C00,
0x0000, 0x0000,  // z
            0x1800, 0x1000, 0x1000, 0x1000, 0x2000, 0x2000, 0x1000, 0x1000,
0x1000, 0x1800,  // {
            0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000, 0x1000,
0x1000, 0x1000,  // |
            0x3000, 0x1000, 0x1000, 0x1000, 0x0800, 0x0800, 0x1000, 0x1000,
0x1000, 0x3000,  // }
            0x0000, 0x0000, 0x0000, 0x7400, 0x4C00, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000,  // ~
};

static const uint16_t font_medium_data[] = {
            0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
// sp
            0x0000, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00,
0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0000, 0x0C00, 0x0C00, 0x0000, 0x0000, 0x0000,
// !
            0x0000, 0x1B00, 0x1B00, 0x1B00, 0x1B00, 0x1B00, 0x0000, 0x0000,
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
// "
             0x0000, 0x1980, 0x1980, 0x1980, 0x1980, 0x7FC0, 0x7FC0, 0x1980,
0x3300, 0x7FC0, 0x7FC0, 0x3300, 0x3300, 0x3300, 0x3300, 0x0000, 0x0000, 0x0000,
// #
             0x0000, 0x1E00, 0x3F00, 0x7580, 0x6580, 0x7400, 0x3C00, 0x1E00,
0x0700, 0x0580, 0x6580, 0x6580, 0x7580, 0x3F00, 0x1E00, 0x0400, 0x0400, 0x0000,
// $
             0x0000, 0x7000, 0xD800, 0xD840, 0xD8C0, 0xD980, 0x7300, 0x0600,
0x0C00, 0x1B80, 0x36C0, 0x66C0, 0x46C0, 0x06C0, 0x0380, 0x0000, 0x0000, 0x0000,
// %
             0x0000, 0x1E00, 0x3F00, 0x3300, 0x3300, 0x3300, 0x1E00, 0x0C00,
0x3CC0, 0x66C0, 0x6380, 0x6180, 0x6380, 0x3EC0, 0x1C80, 0x0000, 0x0000, 0x0000,
// &
             0x0000, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0000, 0x0000,
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
// '
             0x0080, 0x0100, 0x0300, 0x0600, 0x0600, 0x0400, 0x0C00, 0x0C00,
0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0400, 0x0600, 0x0600, 0x0300, 0x0100, 0x0080,
// (
             0x2000, 0x1000, 0x1800, 0x0C00, 0x0C00, 0x0400, 0x0600, 0x0600,
0x0600, 0x0600, 0x0600, 0x0600, 0x0400, 0x0C00, 0x0C00, 0x1800, 0x1000, 0x2000,
// )
             0x0000, 0x0C00, 0x2D00, 0x3F00, 0x1E00, 0x3300, 0x0000, 0x0000,
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
// *
             0x0000, 0x0000, 0x0000, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0xFFC0,
0xFFC0, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
// +
             0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0C00, 0x0C00, 0x0400, 0x0400, 0x0800,
// ,
             0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x1E00, 0x1E00, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
// -
             0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0C00, 0x0C00, 0x0000, 0x0000, 0x0000,
// .
             0x0000, 0x0300, 0x0300, 0x0300, 0x0600, 0x0600, 0x0600, 0x0600,
0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x1800, 0x1800, 0x1800, 0x0000, 0x0000, 0x0000,
// /
             0x0000, 0x1E00, 0x3F00, 0x3300, 0x6180, 0x6180, 0x6180, 0x6D80,
0x6D80, 0x6180, 0x6180, 0x6180, 0x3300, 0x3F00, 0x1E00, 0x0000, 0x0000, 0x0000,
// 0
             0x0000, 0x0600, 0x0E00, 0x1E00, 0x3600, 0x2600, 0x0600, 0x0600,
0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0000, 0x0000, 0x0000,
// 1
             0x0000, 0x1E00, 0x3F00, 0x7380, 0x6180, 0x6180, 0x0180, 0x0300,
0x0600, 0x0C00, 0x1800, 0x3000, 0x6000, 0x7F80, 0x7F80, 0x0000, 0x0000, 0x0000,
// 2
             0x0000, 0x1C00, 0x3E00, 0x6300, 0x6300, 0x0300, 0x0E00, 0x0E00,
0x0300, 0x0180, 0x0180, 0x6180, 0x7380, 0x3F00, 0x1E00, 0x0000, 0x0000, 0x0000,
// 3
             0x0000, 0x0600, 0x0E00, 0x0E00, 0x1E00, 0x1E00, 0x1600, 0x3600,
0x3600, 0x6600, 0x7F80, 0x7F80, 0x0600, 0x0600, 0x0600, 0x0000, 0x0000, 0x0000,
// 4
            0x0000, 0x7F00, 0x7F00, 0x6000, 0x6000, 0x6000, 0x6E00, 0x7F00,
0x6380, 0x0180, 0x0180, 0x6180, 0x7380, 0x3F00, 0x1E00, 0x0000, 0x0000, 0x0000,
// 5
            0x0000, 0x1E00, 0x3F00, 0x3380, 0x6180, 0x6000, 0x6E00, 0x7F00,
0x7380, 0x6180, 0x6180, 0x6180, 0x3380, 0x3F00, 0x1E00, 0x0000, 0x0000, 0x0000,
// 6
            0x0000, 0x7F80, 0x7F80, 0x0180, 0x0300, 0x0300, 0x0600, 0x0600,
0x0C00, 0x0C00, 0x0C00, 0x0800, 0x1800, 0x1800, 0x1800, 0x0000, 0x0000, 0x0000,
// 7
            0x0000, 0x1E00, 0x3F00, 0x6380, 0x6180, 0x6180, 0x2100, 0x1E00,
0x3F00, 0x6180, 0x6180, 0x6180, 0x6180, 0x3F00, 0x1E00, 0x0000, 0x0000, 0x0000,
// 8
            0x0000, 0x1E00, 0x3F00, 0x7300, 0x6180, 0x6180, 0x6180, 0x7380,
0x3F80, 0x1D80, 0x0180, 0x6180, 0x7300, 0x3F00, 0x1E00, 0x0000, 0x0000, 0x0000,
// 9
            0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0C00, 0x0C00, 0x0000,
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0C00, 0x0C00, 0x0000, 0x0000, 0x0000,
// :
            0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0C00, 0x0C00,
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0C00, 0x0C00, 0x0400, 0x0400, 0x0800,
// ;
            0x0000, 0x0000, 0x0000, 0x0000, 0x0080, 0x0380, 0x0E00, 0x3800,
0x6000, 0x3800, 0x0E00, 0x0380, 0x0080, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
// <
            0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x7F80, 0x7F80, 0x0000,
0x0000, 0x7F80, 0x7F80, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
// =
            0x0000, 0x0000, 0x0000, 0x0000, 0x4000, 0x7000, 0x1C00, 0x0700,
0x0180, 0x0700, 0x1C00, 0x7000, 0x4000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
// >
            0x0000, 0x1F00, 0x3F80, 0x71C0, 0x60C0, 0x00C0, 0x01C0, 0x0380,
0x0700, 0x0E00, 0x0C00, 0x0C00, 0x0000, 0x0C00, 0x0C00, 0x0000, 0x0000, 0x0000,
// ?
            0x0000, 0x1E00, 0x3F00, 0x3180, 0x7180, 0x6380, 0x6F80, 0x6D80,
0x6D80, 0x6F80, 0x6780, 0x6000, 0x3200, 0x3E00, 0x1C00, 0x0000, 0x0000, 0x0000,
// @
            0x0000, 0x0E00, 0x0E00, 0x1B00, 0x1B00, 0x1B00, 0x1B00, 0x3180,
0x3180, 0x3F80, 0x3F80, 0x3180, 0x60C0, 0x60C0, 0x60C0, 0x0000, 0x0000, 0x0000,
// A
            0x0000, 0x7C00, 0x7E00, 0x6300, 0x6300, 0x6300, 0x6300, 0x7E00,
0x7E00, 0x6300, 0x6180, 0x6180, 0x6380, 0x7F00, 0x7E00, 0x0000, 0x0000, 0x0000,
// B
            0x0000, 0x1E00, 0x3F00, 0x3180, 0x6180, 0x6000, 0x6000, 0x6000,
0x6000, 0x6000, 0x6000, 0x6180, 0x3180, 0x3F00, 0x1E00, 0x0000, 0x0000, 0x0000,
// C
            0x0000, 0x7C00, 0x7F00, 0x6300, 0x6380, 0x6180, 0x6180, 0x6180,
0x6180, 0x6180, 0x6180, 0x6300, 0x6300, 0x7E00, 0x7C00, 0x0000, 0x0000, 0x0000,
// D
            0x0000, 0x7F80, 0x7F80, 0x6000, 0x6000, 0x6000, 0x6000, 0x7F00,
0x7F00, 0x6000, 0x6000, 0x6000, 0x6000, 0x7F80, 0x7F80, 0x0000, 0x0000, 0x0000,
// E
            0x0000, 0x7F80, 0x7F80, 0x6000, 0x6000, 0x6000, 0x6000, 0x7F00,
0x7F00, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x0000, 0x0000, 0x0000,
// F
```

```
            0x0000, 0x1E00, 0x3F00, 0x3180, 0x6180, 0x6000, 0x6000, 0x6000,
0x6380, 0x6380, 0x6180, 0x6180, 0x3180, 0x3F80, 0x1E00, 0x0000, 0x0000, 0x0000,
// G
            0x0000, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x7F80,
0x7F80, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x0000, 0x0000, 0x0000,
// H
            0x0000, 0x3F00, 0x3F00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00,
0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x3F00, 0x3F00, 0x0000, 0x0000, 0x0000,
// I
            0x0000, 0x0180, 0x0180, 0x0180, 0x0180, 0x0180, 0x0180, 0x0180,
0x0180, 0x0180, 0x6180, 0x6180, 0x7380, 0x3F00, 0x1E00, 0x0000, 0x0000, 0x0000,
// J
            0x0000, 0x60C0, 0x6180, 0x6300, 0x6600, 0x6600, 0x6C00, 0x7800,
0x7C00, 0x6600, 0x6600, 0x6300, 0x6180, 0x6180, 0x60C0, 0x0000, 0x0000, 0x0000,
// K
            0x0000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000,
0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x7F80, 0x7F80, 0x0000, 0x0000, 0x0000,
// L
            0x0000, 0x71C0, 0x71C0, 0x7BC0, 0x7AC0, 0x6AC0, 0x6AC0, 0x6EC0,
0x64C0, 0x60C0, 0x60C0, 0x60C0, 0x60C0, 0x60C0, 0x60C0, 0x0000, 0x0000, 0x0000,
// M
            0x0000, 0x7180, 0x7180, 0x7980, 0x7980, 0x7980, 0x6D80, 0x6D80,
0x6D80, 0x6580, 0x6780, 0x6780, 0x6780, 0x6380, 0x6380, 0x0000, 0x0000, 0x0000,
// N
            0x0000, 0x1E00, 0x3F00, 0x3300, 0x6180, 0x6180, 0x6180, 0x6180,
0x6180, 0x6180, 0x6180, 0x6180, 0x3300, 0x3F00, 0x1E00, 0x0000, 0x0000, 0x0000,
// O
            0x0000, 0x7E00, 0x7F00, 0x6380, 0x6180, 0x6180, 0x6180, 0x6380,
0x7F00, 0x7E00, 0x6000, 0x6000, 0x6000, 0x6000, 0x6000, 0x0000, 0x0000, 0x0000,
// P
            0x0000, 0x1E00, 0x3F00, 0x3300, 0x6180, 0x6180, 0x6180, 0x6180,
0x6180, 0x6180, 0x6580, 0x6780, 0x3300, 0x3F80, 0x1E40, 0x0000, 0x0000, 0x0000,
// Q
            0x0000, 0x7E00, 0x7F00, 0x6380, 0x6180, 0x6180, 0x6380, 0x7F00,
0x7E00, 0x6600, 0x6300, 0x6300, 0x6180, 0x6180, 0x60C0, 0x0000, 0x0000, 0x0000,
// R
            0x0000, 0x0E00, 0x1F00, 0x3180, 0x3180, 0x3000, 0x3800, 0x1E00,
0x0700, 0x0380, 0x6180, 0x6180, 0x3180, 0x3F00, 0x1E00, 0x0000, 0x0000, 0x0000,
// S
            0x0000, 0xFFC0, 0xFFC0, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00,
0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0000, 0x0000, 0x0000,
// T
            0x0000, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180,
0x6180, 0x6180, 0x6180, 0x6180, 0x7380, 0x3F00, 0x1E00, 0x0000, 0x0000, 0x0000,
// U
            0x0000, 0x60C0, 0x60C0, 0x60C0, 0x3180, 0x3180, 0x3180, 0x1B00,
0x1B00, 0x1B00, 0x1B00, 0x0E00, 0x0E00, 0x0E00, 0x0400, 0x0000, 0x0000, 0x0000,
// V
            0x0000, 0xC0C0, 0xC0C0, 0xC0C0, 0xC0C0, 0xC0C0, 0xCCC0, 0x4C80,
0x4C80, 0x5E80, 0x5280, 0x5280, 0x7380, 0x6180, 0x6180, 0x0000, 0x0000, 0x0000,
// W
            0x0000, 0xC0C0, 0x6080, 0x6180, 0x3300, 0x3B00, 0x1E00, 0x0C00,
0x0C00, 0x1E00, 0x1F00, 0x3B00, 0x7180, 0x6180, 0xC0C0, 0x0000, 0x0000, 0x0000,
// X
            0x0000, 0xC0C0, 0x6180, 0x6180, 0x3300, 0x3300, 0x1E00, 0x1E00,
0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0000, 0x0000, 0x0000,
// Y
            0x0000, 0x3F80, 0x3F80, 0x0180, 0x0300, 0x0300, 0x0600, 0x0C00,
0x0C00, 0x1800, 0x1800, 0x3000, 0x6000, 0x7F80, 0x7F80, 0x0000, 0x0000, 0x0000,
// Z
            0x0F00, 0x0F00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00,
0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0F00, 0x0F00,
// [
            0x0000, 0x1800, 0x1800, 0x1800, 0x0C00, 0x0C00, 0x0C00, 0x0C00,
0x0600, 0x0600, 0x0600, 0x0600, 0x0300, 0x0300, 0x0300, 0x0000, 0x0000, 0x0000,
/* \ */
            0x1E00, 0x1E00, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600,
0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x1E00, 0x1E00,
// ]
            0x0000, 0x0C00, 0x0C00, 0x1E00, 0x1200, 0x3300, 0x3300, 0x6180,
0x6180, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
// ^
            0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0xFFE0, 0x0000,
// _
            0x0000, 0x3800, 0x1800, 0x0C00, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
// `
            0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x1F00, 0x3F80, 0x6180,
0x0180, 0x1F80, 0x3F80, 0x6180, 0x6380, 0x7F80, 0x38C0, 0x0000, 0x0000, 0x0000,
// a
            0x0000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6E00, 0x7F00, 0x7380,
0x6180, 0x6180, 0x6180, 0x6180, 0x7380, 0x7F00, 0x6E00, 0x0000, 0x0000, 0x0000,
// b
            0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x1E00, 0x3F00, 0x7380,
0x6180, 0x6000, 0x6000, 0x6180, 0x7380, 0x3F00, 0x1E00, 0x0000, 0x0000, 0x0000,
// c
            0x0000, 0x0180, 0x0180, 0x0180, 0x0180, 0x1D80, 0x3F80, 0x7380,
0x6180, 0x6180, 0x6180, 0x6180, 0x7380, 0x3F80, 0x1D80, 0x0000, 0x0000, 0x0000,
// d
            0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x1E00, 0x3F00, 0x7300,
0x6180, 0x7F80, 0x7F80, 0x6000, 0x7180, 0x3F00, 0x1E00, 0x0000, 0x0000, 0x0000,
// e
            0x0000, 0x07C0, 0x0FC0, 0x0C00, 0x0C00, 0x7F80, 0x7F80, 0x0C00,
0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0000, 0x0000, 0x0000,
// f
            0x0000, 0x0000, 0x0000, 0x0000, 0x1D80, 0x3F80, 0x7380, 0x6180,
0x6180, 0x6180, 0x6180, 0x7380, 0x3F80, 0x1D80, 0x0180, 0x6380, 0x7F00, 0x3E00,
// g
            0x0000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6F00, 0x7F80, 0x7180,
0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x0000, 0x0000, 0x0000,
// h
            0x0000, 0x0600, 0x0600, 0x0000, 0x0000, 0x3E00, 0x3E00, 0x0600,
0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0000, 0x0000, 0x0000,
// i
            0x0600, 0x0600, 0x0000, 0x0000, 0x3E00, 0x3E00, 0x0600, 0x0600,
0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x4600, 0x7E00, 0x3C00,
// j
            0x0000, 0x6000, 0x6000, 0x6000, 0x6000, 0x6180, 0x6300, 0x6600,
0x6C00, 0x7C00, 0x7600, 0x6300, 0x6300, 0x6180, 0x60C0, 0x0000, 0x0000, 0x0000,
// k
          0x0000, 0x3E00, 0x3E00, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600,
0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0000, 0x0000, 0x0000,
// l
          0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0xDD80, 0xFFC0, 0xCEC0,
0xCCC0, 0xCCC0, 0xCCC0, 0xCCC0, 0xCCC0, 0xCCC0, 0xCCC0, 0x0000, 0x0000, 0x0000,
// m
          0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x6F00, 0x7F80, 0x7180,
0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x6180, 0x0000, 0x0000, 0x0000,
// n
          0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x1E00, 0x3F00, 0x7380,
0x6180, 0x6180, 0x6180, 0x6180, 0x7380, 0x3F00, 0x1E00, 0x0000, 0x0000, 0x0000,
// o
          0x0000, 0x0000, 0x0000, 0x0000, 0x6E00, 0x7F00, 0x7380, 0x6180,
0x6180, 0x6180, 0x6180, 0x7380, 0x7F00, 0x6E00, 0x6000, 0x6000, 0x6000, 0x6000,
// p
          0x0000, 0x0000, 0x0000, 0x0000, 0x1D80, 0x3F80, 0x7380, 0x6180,
0x6180, 0x6180, 0x6180, 0x7380, 0x3F80, 0x1D80, 0x0180, 0x0180, 0x0180, 0x0180,
// q
          0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x6700, 0x3F80, 0x3900,
0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x3000, 0x0000, 0x0000, 0x0000,
// r
          0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x1E00, 0x3F80, 0x6180,
0x6000, 0x7F00, 0x3F80, 0x0180, 0x6180, 0x7F00, 0x1E00, 0x0000, 0x0000, 0x0000,
// s
          0x0000, 0x0000, 0x0800, 0x1800, 0x1800, 0x7F00, 0x7F00, 0x1800,
0x1800, 0x1800, 0x1800, 0x1800, 0x1800, 0x1F80, 0x0F80, 0x0000, 0x0000, 0x0000,
// t
          0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x6180, 0x6180, 0x6180,
0x6180, 0x6180, 0x6180, 0x6180, 0x6380, 0x7F80, 0x3D80, 0x0000, 0x0000, 0x0000,
// u
          0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x60C0, 0x3180, 0x3180,
0x3180, 0x1B00, 0x1B00, 0x1B00, 0x0E00, 0x0E00, 0x0600, 0x0000, 0x0000, 0x0000,
// v
          0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0xDD80, 0xDD80, 0xDD80,
0x5500, 0x5500, 0x5500, 0x7700, 0x7700, 0x2200, 0x2200, 0x0000, 0x0000, 0x0000,
// w
          0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x6180, 0x3300, 0x3300,
0x1E00, 0x0C00, 0x0C00, 0x1E00, 0x3300, 0x3300, 0x6180, 0x0000, 0x0000, 0x0000,
// x
          0x0000, 0x0000, 0x0000, 0x0000, 0x6180, 0x6180, 0x3180, 0x3300,
0x3300, 0x1B00, 0x1B00, 0x1B00, 0x0E00, 0x0E00, 0x0E00, 0x1C00, 0x7C00, 0x7000,
// y
          0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x7FC0, 0x7FC0, 0x0180,
0x0300, 0x0600, 0x0C00, 0x1800, 0x3000, 0x7FC0, 0x7FC0, 0x0000, 0x0000, 0x0000,
// z
          0x0380, 0x0780, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0E00,
0x1C00, 0x1C00, 0x0E00, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0780, 0x0380,
// {
          0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600,
0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600, 0x0600,
// |
          0x3800, 0x3C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0E00,
0x0700, 0x0700, 0x0E00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x0C00, 0x3C00, 0x3800,
// }
```

```
          0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x3880,
0x7F80, 0x4700, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
// ~
};

static const uint16_t font_large_data[] = {

0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x
0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [ ]

0x03E0,0x03E0,0x03E0,0x03E0,0x03E0,0x03E0,0x03E0,0x03E0,0x03C0,0x03C0,0x01C0,0x
01C0,0x01C0,0x01C0,0x01C0,0x0000,0x0000,0x0000,0x03E0,0x03E0,0x03E0,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [!]

0x1E3C,0x1E3C,0x1E3C,0x1E3C,0x1E3C,0x1E3C,0x1E3C,0x0000,0x0000,0x0000,0x0000,0x
0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = ["]

0x01CE,0x03CE,0x03DE,0x039E,0x039C,0x079C,0x3FFF,0x7FFF,0x0738,0x0F38,0x0F78,0x
0F78,0x0E78,0xFFFF,0xFFFF,0x1EF0,0x1CF0,0x1CE0,0x3CE0,0x3DE0,0x39E0,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [#]

0x03FC,0x0FFE,0x1FEE,0x1EE0,0x1EE0,0x1EE0,0x1EE0,0x1FE0,0x0FE0,0x07E0,0x03F0,0x
01FC,0x01FE,0x01FE,0x01FE,0x01FE,0x01FE,0x01FE,0x3DFE,0x3FFC,0x0FF0,0x01E0,0x01
E0,0x0000,0x0000,0x0000, // Ascii = [$]

0x3E03,0xF707,0xE78F,0xE78E,0xE39E,0xE3BC,0xE7B8,0xE7F8,0xF7F0,0x3FE0,0x01C0,0x
03FF,0x07FF,0x07F3,0x0FF3,0x1EF3,0x3CF3,0x38F3,0x78F3,0xF07F,0xE03F,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [%]

0x07E0,0x0FF8,0x0F78,0x1F78,0x1F78,0x1F78,0x0F78,0x0FF0,0x0FE0,0x1F80,0x7FC3,0x
FBC3,0xF3E7,0xF1F7,0xF0F7,0xF0FF,0xF07F,0xF83E,0x7C7F,0x3FFF,0x1FEF,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [&]

0x03E0,0x03E0,0x03E0,0x03E0,0x03E0,0x03C0,0x01C0,0x0000,0x0000,0x0000,0x0000,0x
0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [']

0x003F,0x007C,0x01F0,0x01E0,0x03C0,0x07C0,0x0780,0x0780,0x0F80,0x0F00,0x0F00,0x
0F00,0x0F00,0x0F00,0x0F00,0x0F80,0x0780,0x0780,0x07C0,0x03C0,0x01E0,0x01F0,0x00
7C,0x003F,0x000F,0x0000, // Ascii = [(]

0x7E00,0x1F00,0x07C0,0x03C0,0x01E0,0x01F0,0x00F0,0x00F0,0x00F8,0x0078,0x0078,0x
0078,0x0078,0x0078,0x0078,0x00F8,0x00F0,0x00F0,0x01F0,0x01E0,0x03C0,0x07C0,0x1F
00,0x7E00,0x7800,0x0000, // Ascii = [)]

0x03E0,0x03C0,0x01C0,0x39CE,0x3FFF,0x3F7F,0x0320,0x0370,0x07F8,0x0F78,0x1F3C,0x
0638,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [*]

0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x01C0,0x01C0,0x01C0,0x01C0,0x01C0,0x
01C0,0x01C0,0xFFFF,0xFFFF,0x01C0,0x01C0,0x01C0,0x01C0,0x01C0,0x01C0,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [+]
```

```
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x
0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x03E0,0x03E0,0x03E0,0x03E0,0x01E0,0x01
E0,0x01E0,0x01C0,0x0380, // Ascii = [,]

0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x
3FFE,0x3FFE,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [-]

0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x
0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x03E0,0x03E0,0x03E0,0x03E0,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [.]

0x000F,0x000F,0x001E,0x001E,0x003C,0x003C,0x0078,0x0078,0x00F0,0x00F0,0x01E0,0x
01E0,0x03C0,0x03C0,0x0780,0x0780,0x0F00,0x0F00,0x1E00,0x1E00,0x3C00,0x3C00,0x78
00,0x7800,0xF000,0x0000, // Ascii = [/]

0x07F0,0x0FF8,0x1F7C,0x3E3E,0x3C1E,0x7C1F,0x7C1F,0x780F,0x780F,0x780F,0x780F,0x
780F,0x780F,0x780F,0x7C1F,0x7C1F,0x3C1E,0x3E3E,0x1F7C,0x0FF8,0x07F0,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [0]

0x00F0,0x07F0,0x3FF0,0x3FF0,0x01F0,0x01F0,0x01F0,0x01F0,0x01F0,0x01F0,0x01F0,0x
01F0,0x01F0,0x01F0,0x01F0,0x01F0,0x01F0,0x01F0,0x01F0,0x3FFF,0x3FFF,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [1]

0x0FE0,0x3FF8,0x3C7C,0x003C,0x003E,0x003E,0x003E,0x003C,0x003C,0x007C,0x00F8,0x
01F0,0x03E0,0x07C0,0x0780,0x0F00,0x1E00,0x3E00,0x3C00,0x3FFE,0x3FFE,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [2]

0x0FF0,0x1FF8,0x1C7C,0x003E,0x003E,0x003E,0x003C,0x003C,0x00F8,0x0FF0,0x0FF8,0x
007C,0x003E,0x001E,0x001E,0x001E,0x001E,0x003E,0x1C7C,0x1FF8,0x1FE0,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [3]

0x0078,0x00F8,0x00F8,0x01F8,0x03F8,0x07F8,0x07F8,0x0F78,0x1E78,0x1E78,0x3C78,0x
7878,0x7878,0xFFFF,0xFFFF,0x0078,0x0078,0x0078,0x0078,0x0078,0x0078,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [4]

0x1FFC,0x1FFC,0x1FFC,0x1E00,0x1E00,0x1E00,0x1E00,0x1E00,0x1FE0,0x1FF8,0x00FC,0x
007C,0x003E,0x003E,0x001E,0x003E,0x003E,0x003C,0x1C7C,0x1FF8,0x1FE0,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [5]

0x01FC,0x07FE,0x0F8E,0x1F00,0x1E00,0x3E00,0x3C00,0x3C00,0x3DF8,0x3FFC,0x7F3E,0x
7E1F,0x3C0F,0x3C0F,0x3C0F,0x3C0F,0x3E0F,0x1E1F,0x1F3E,0x0FFC,0x03F0,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [6]

0x3FFF,0x3FFF,0x3FFF,0x000F,0x001E,0x001E,0x003C,0x0038,0x0078,0x00F0,0x00F0,0x
01E0,0x01E0,0x03C0,0x03C0,0x0780,0x0F80,0x0F80,0x0F00,0x1F00,0x1F00,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [7]

0x07F8,0x0FFC,0x1F3E,0x1E1E,0x3E1E,0x3E1E,0x1E1E,0x1F3C,0x0FF8,0x07F0,0x0FF8,0x
1EFC,0x3E3E,0x3C1F,0x7C1F,0x7C0F,0x7C0F,0x3C1F,0x3F3E,0x1FFC,0x07F0,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [8]

0x07F0,0x0FF8,0x1E7C,0x3C3E,0x3C1E,0x7C1F,0x7C1F,0x7C1F,0x7C1F,0x3C1F,0x3E3F,0x
1FFF,0x07EF,0x001F,0x001E,0x001E,0x003E,0x003C,0x38F8,0x3FF0,0x1FE0,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [9]
```

```
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x03E0,0x03E0,0x03E0,0x03E0,0x0000,0x
0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x03E0,0x03E0,0x03E0,0x03E0,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [:]

0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x03E0,0x03E0,0x03E0,0x03E0,0x0000,0x
0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x03E0,0x03E0,0x03E0,0x03E0,0x01E0,0x01
E0,0x01E0,0x03C0,0x0380, // Ascii = [;]

0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0003,0x000F,0x003F,0x00FC,0x03F0,0x
0FC0,0x3F00,0xFE00,0x3F00,0x0FC0,0x03F0,0x00FC,0x003F,0x000F,0x0003,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [<]

0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0xFFFF,0x
FFFF,0x0000,0x0000,0x0000,0xFFFF,0xFFFF,0x0000,0x0000,0x0000,0x0000,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [=]

0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0xE000,0xF800,0x7E00,0x1F80,0x07E0,0x
01F8,0x007E,0x001F,0x007E,0x01F8,0x07E0,0x1F80,0x7E00,0xF800,0xE000,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [>]

0x1FF0,0x3FFC,0x383E,0x381F,0x381F,0x001E,0x001E,0x003C,0x0078,0x00F0,0x01E0,0x
03C0,0x03C0,0x07C0,0x07C0,0x0000,0x0000,0x0000,0x07C0,0x07C0,0x07C0,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [?]

0x03F8,0x0FFE,0x1F1E,0x3E0F,0x3C7F,0x78FF,0x79EF,0x73C7,0xF3C7,0xF38F,0xF38F,0x
F38F,0xF39F,0xF39F,0x73FF,0x7BFF,0x79F7,0x3C00,0x1F1C,0x0FFC,0x03F8,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [@]

0x0000,0x0000,0x0000,0x03E0,0x03E0,0x07F0,0x07F0,0x07F0,0x0F78,0x0F78,0x0E7C,0x
1E3C,0x1E3C,0x3C3E,0x3FFE,0x3FFF,0x781F,0x780F,0xF00F,0xF007,0xF007,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [A]

0x0000,0x0000,0x0000,0x3FF8,0x3FFC,0x3C3E,0x3C1E,0x3C1E,0x3C1E,0x3C3E,0x3C7C,0x
3FF0,0x3FF8,0x3C7E,0x3C1F,0x3C1F,0x3C0F,0x3C0F,0x3C1F,0x3FFE,0x3FF8,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [B]

0x0000,0x0000,0x0000,0x01FF,0x07FF,0x1F87,0x3E00,0x3C00,0x7C00,0x7800,0x7800,0x
7800,0x7800,0x7800,0x7C00,0x7C00,0x3E00,0x3F00,0x1F83,0x07FF,0x01FF,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [C]

0x0000,0x0000,0x0000,0x7FF0,0x7FFC,0x787E,0x781F,0x781F,0x780F,0x780F,0x780F,0x
780F,0x780F,0x780F,0x780F,0x780F,0x781F,0x781E,0x787E,0x7FF8,0x7FE0,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [D]

0x0000,0x0000,0x0000,0x3FFF,0x3FFF,0x3E00,0x3E00,0x3E00,0x3E00,0x3E00,0x3E00,0x
3FFE,0x3FFE,0x3E00,0x3E00,0x3E00,0x3E00,0x3E00,0x3E00,0x3FFF,0x3FFF,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [E]

0x0000,0x0000,0x0000,0x1FFF,0x1FFF,0x1E00,0x1E00,0x1E00,0x1E00,0x1E00,0x1E00,0x
1FFF,0x1FFF,0x1E00,0x1E00,0x1E00,0x1E00,0x1E00,0x1E00,0x1E00,0x1E00,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [F]

0x0000,0x0000,0x0000,0x03FE,0x0FFF,0x1F87,0x3E00,0x7C00,0x7C00,0x7800,0xF800,0x
F800,0xF87F,0xF87F,0x780F,0x7C0F,0x7C0F,0x3E0F,0x1F8F,0x0FFF,0x03FE,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [G]
```

```
0x0000,0x0000,0x0000,0x7C1F,0x7C1F,0x7C1F,0x7C1F,0x7C1F,0x7C1F,0x7C1F,0x7C1F,0x
7FFF,0x7FFF,0x7C1F,0x7C1F,0x7C1F,0x7C1F,0x7C1F,0x7C1F,0x7C1F,0x7C1F,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [H]

0x0000,0x0000,0x0000,0x3FFF,0x3FFF,0x03E0,0x03E0,0x03E0,0x03E0,0x03E0,0x03E0,0x
03E0,0x03E0,0x03E0,0x03E0,0x03E0,0x03E0,0x03E0,0x03E0,0x3FFF,0x3FFF,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [I]

0x0000,0x0000,0x0000,0x1FFC,0x1FFC,0x007C,0x007C,0x007C,0x007C,0x007C,0x007C,0x
007C,0x007C,0x007C,0x007C,0x007C,0x0078,0x0078,0x38F8,0x3FF0,0x3FC0,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [J]

0x0000,0x0000,0x0000,0x3C1F,0x3C1E,0x3C3C,0x3C78,0x3CF0,0x3DE0,0x3FE0,0x3FC0,0x
3F80,0x3FC0,0x3FE0,0x3DF0,0x3CF0,0x3C78,0x3C7C,0x3C3E,0x3C1F,0x3C0F,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [K]

0x0000,0x0000,0x0000,0x3E00,0x3E00,0x3E00,0x3E00,0x3E00,0x3E00,0x3E00,0x3E00,0x
3E00,0x3E00,0x3E00,0x3E00,0x3E00,0x3E00,0x3E00,0x3E00,0x3FFF,0x3FFF,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [L]

0x0000,0x0000,0x0000,0xF81F,0xFC1F,0xFC1F,0xFE3F,0xFE3F,0xFE3F,0xFF7F,0xFF77,0x
FF77,0xF7F7,0xF7E7,0xF3E7,0xF3E7,0xF3C7,0xF007,0xF007,0xF007,0xF007,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [M]

0x0000,0x0000,0x0000,0x7C0F,0x7C0F,0x7E0F,0x7F0F,0x7F0F,0x7F8F,0x7F8F,0x7FCF,0x
7BEF,0x79EF,0x79FF,0x78FF,0x78FF,0x787F,0x783F,0x783F,0x781F,0x781F,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [N]

0x0000,0x0000,0x0000,0x07F0,0x1FFC,0x3E3E,0x7C1F,0x780F,0x780F,0xF80F,0xF80F,0x
F80F,0xF80F,0xF80F,0xF80F,0x780F,0x780F,0x7C1F,0x3E3E,0x1FFC,0x07F0,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [O]

0x0000,0x0000,0x0000,0x3FFC,0x3FFF,0x3E1F,0x3E0F,0x3E0F,0x3E0F,0x3E0F,0x3E1F,0x
3E3F,0x3FFC,0x3FF0,0x3E00,0x3E00,0x3E00,0x3E00,0x3E00,0x3E00,0x3E00,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [P]

0x0000,0x0000,0x0000,0x07F0,0x1FFC,0x3E3E,0x7C1F,0x780F,0x780F,0xF80F,0xF80F,0x
F80F,0xF80F,0xF80F,0xF80F,0x780F,0x780F,0x7C1F,0x3E3E,0x1FFC,0x07F8,0x007C,0x00
3F,0x000F,0x0003,0x0000, // Ascii = [Q]

0x0000,0x0000,0x0000,0x3FF0,0x3FFC,0x3C7E,0x3C3E,0x3C1E,0x3C1E,0x3C3E,0x3C3C,0x
3CFC,0x3FF0,0x3FE0,0x3DF0,0x3CF8,0x3C7C,0x3C3E,0x3C1E,0x3C1F,0x3C0F,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [R]

0x0000,0x0000,0x0000,0x07FC,0x1FFE,0x3E0E,0x3C00,0x3C00,0x3C00,0x3E00,0x1FC0,0x
0FF8,0x03FE,0x007F,0x001F,0x000F,0x000F,0x201F,0x3C3E,0x3FFC,0x1FF0,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [S]

0x0000,0x0000,0x0000,0xFFFF,0xFFFF,0x03E0,0x03E0,0x03E0,0x03E0,0x03E0,0x03E0,0x
03E0,0x03E0,0x03E0,0x03E0,0x03E0,0x03E0,0x03E0,0x03E0,0x03E0,0x03E0,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [T]

0x0000,0x0000,0x0000,0x7C0F,0x7C0F,0x7C0F,0x7C0F,0x7C0F,0x7C0F,0x7C0F,0x7C0F,0x
7C0F,0x7C0F,0x7C0F,0x7C0F,0x7C0F,0x3C1E,0x3C1E,0x3E3E,0x1FFC,0x07F0,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [U]
```

```
0x0000,0x0000,0x0000,0xF007,0xF007,0xF807,0x780F,0x7C0F,0x3C1E,0x3C1E,0x3E1E,0x
1E3C,0x1F3C,0x1F78,0x0F78,0x0FF8,0x07F0,0x07F0,0x07F0,0x03E0,0x03E0,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [V]

0x0000,0x0000,0x0000,0xE003,0xF003,0xF003,0xF007,0xF3E7,0xF3E7,0xF3E7,0x73E7,0x
7BF7,0x7FF7,0x7FFF,0x7F7F,0x7F7F,0x7F7E,0x3F7E,0x3E3E,0x3E3E,0x3E3E,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [W]

0x0000,0x0000,0x0000,0xF807,0x7C0F,0x3E1E,0x3E3E,0x1F3C,0x0FF8,0x07F0,0x07E0,0x
03E0,0x03E0,0x07F0,0x0FF8,0x0F7C,0x1E7C,0x3C3E,0x781F,0x780F,0xF00F,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [X]

0x0000,0x0000,0x0000,0xF807,0x7807,0x7C0F,0x3C1E,0x3E1E,0x1F3C,0x0F78,0x0FF8,0x
07F0,0x03E0,0x03E0,0x03E0,0x03E0,0x03E0,0x03E0,0x03E0,0x03E0,0x03E0,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [Y]

0x0000,0x0000,0x0000,0x7FFF,0x7FFF,0x000F,0x001F,0x003E,0x007C,0x00F8,0x00F0,0x
01E0,0x03E0,0x07C0,0x0F80,0x0F00,0x1E00,0x3E00,0x7C00,0x7FFF,0x7FFF,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [Z]

0x07FF,0x0780,0x0780,0x0780,0x0780,0x0780,0x0780,0x0780,0x0780,0x0780,0x0780,0x
0780,0x0780,0x0780,0x0780,0x0780,0x0780,0x0780,0x0780,0x0780,0x0780,0x0780,0x07
80,0x07FF,0x07FF,0x0000, // Ascii = [[]

0x7800,0x7800,0x3C00,0x3C00,0x1E00,0x1E00,0x0F00,0x0F00,0x0780,0x0780,0x03C0,0x
03C0,0x01E0,0x01E0,0x00F0,0x00F0,0x0078,0x0078,0x003C,0x003C,0x001E,0x001E,0x00
0F,0x000F,0x0007,0x0000, // Ascii = [\]

0x7FF0,0x00F0,0x00F0,0x00F0,0x00F0,0x00F0,0x00F0,0x00F0,0x00F0,0x00F0,0x00F0,0x
00F0,0x00F0,0x00F0,0x00F0,0x00F0,0x00F0,0x00F0,0x00F0,0x00F0,0x00F0,0x00F0,0x00
F0,0x7FF0,0x7FF0,0x0000, // Ascii = []]

0x00C0,0x01C0,0x01C0,0x03E0,0x03E0,0x07F0,0x07F0,0x0778,0x0F78,0x0F38,0x1E3C,0x
1E3C,0x3C1E,0x3C1E,0x380F,0x780F,0x7807,0x0000,0x0000,0x0000,0x0000,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [^]

0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x
0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0xFFFF,0xFF
FF,0x0000,0x0000,0x0000, // Ascii = [_]

0x00F0,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x
0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [`]

0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0FF8,0x3FFC,0x3C7C,0x003E,0x003E,0x
003E,0x07FE,0x1FFE,0x3E3E,0x7C3E,0x783E,0x7C3E,0x7C7E,0x3FFF,0x1FCF,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [a]

0x3C00,0x3C00,0x3C00,0x3C00,0x3C00,0x3C00,0x3DF8,0x3FFE,0x3F3E,0x3E1F,0x3C0F,0x
3C0F,0x3C0F,0x3C0F,0x3C0F,0x3C0F,0x3C1F,0x3C1E,0x3F3E,0x3FFC,0x3BF0,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [b]

0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x03FE,0x0FFF,0x1F87,0x3E00,0x3E00,0x
3C00,0x7C00,0x7C00,0x7C00,0x3C00,0x3E00,0x3E00,0x1F87,0x0FFF,0x03FE,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [c]
```

```
0x001F,0x001F,0x001F,0x001F,0x001F,0x001F,0x07FF,0x1FFF,0x3E3F,0x3C1F,0x7C1F,0x
7C1F,0x7C1F,0x781F,0x781F,0x7C1F,0x7C1F,0x3C3F,0x3E7F,0x1FFF,0x0FDF,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [d]

0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x03F8,0x0FFC,0x1F3E,0x3E1E,0x3C1F,0x
7C1F,0x7FFF,0x7FFF,0x7C00,0x7C00,0x3C00,0x3E00,0x1F07,0x0FFF,0x03FE,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [e]

0x01FF,0x03E1,0x03C0,0x07C0,0x07C0,0x07C0,0x7FFF,0x7FFF,0x07C0,0x07C0,0x07C0,0x
07C0,0x07C0,0x07C0,0x07C0,0x07C0,0x07C0,0x07C0,0x07C0,0x07C0,0x07C0,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [f]

0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x07EF,0x1FFF,0x3E7F,0x3C1F,0x7C1F,0x
7C1F,0x781F,0x781F,0x781F,0x7C1F,0x7C1F,0x3C3F,0x3E7F,0x1FFF,0x0FDF,0x001E,0x00
1E,0x001E,0x387C,0x3FF8, // Ascii = [g]

0x3C00,0x3C00,0x3C00,0x3C00,0x3C00,0x3C00,0x3DFC,0x3FFE,0x3F9E,0x3F1F,0x3E1F,0x
3C1F,0x3C1F,0x3C1F,0x3C1F,0x3C1F,0x3C1F,0x3C1F,0x3C1F,0x3C1F,0x3C1F,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [h]

0x01F0,0x01F0,0x0000,0x0000,0x0000,0x0000,0x7FE0,0x7FE0,0x01E0,0x01E0,0x01E0,0x
01E0,0x01E0,0x01E0,0x01E0,0x01E0,0x01E0,0x01E0,0x01E0,0x01E0,0x01E0,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [i]

0x00F8,0x00F8,0x0000,0x0000,0x0000,0x0000,0x3FF8,0x3FF8,0x00F8,0x00F8,0x00F8,0x
00F8,0x00F8,0x00F8,0x00F8,0x00F8,0x00F8,0x00F8,0x00F8,0x00F8,0x00F8,0x00F8,0x00
F8,0x00F0,0x71F0,0x7FE0, // Ascii = [j]

0x3C00,0x3C00,0x3C00,0x3C00,0x3C00,0x3C00,0x3C1F,0x3C3E,0x3C7C,0x3CF8,0x3DF0,0x
3DE0,0x3FC0,0x3FC0,0x3FE0,0x3DF0,0x3CF8,0x3C7C,0x3C3E,0x3C1F,0x3C1F,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [k]

0x7FF0,0x01F0,0x01F0,0x01F0,0x01F0,0x01F0,0x01F0,0x01F0,0x01F0,0x01F0,0x01F0,0x
01F0,0x01F0,0x01F0,0x01F0,0x01F0,0x01F0,0x01F0,0x01F0,0x01F0,0x01F0,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [l]

0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0xF79E,0xFFFF,0xFFFF,0xFFFF,0xFBE7,0x
F9E7,0xF1C7,0xF1C7,0xF1C7,0xF1C7,0xF1C7,0xF1C7,0xF1C7,0xF1C7,0xF1C7,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [m]

0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x3DFC,0x3FFE,0x3F9E,0x3F1F,0x3E1F,0x
3C1F,0x3C1F,0x3C1F,0x3C1F,0x3C1F,0x3C1F,0x3C1F,0x3C1F,0x3C1F,0x3C1F,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [n]

0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x07F0,0x1FFC,0x3E3E,0x3C1F,0x7C1F,0x
780F,0x780F,0x780F,0x780F,0x780F,0x7C1F,0x3C1F,0x3E3E,0x1FFC,0x07F0,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [o]

0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x3DF8,0x3FFE,0x3F3E,0x3E1F,0x3C0F,0x
3C0F,0x3C0F,0x3C0F,0x3C0F,0x3C0F,0x3C1F,0x3E1E,0x3F3E,0x3FFC,0x3FF8,0x3C00,0x3C
00,0x3C00,0x3C00,0x3C00, // Ascii = [p]

0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x07EE,0x1FFE,0x3E7E,0x3C1E,0x7C1E,0x
781E,0x781E,0x781E,0x781E,0x781E,0x7C1E,0x7C3E,0x3E7E,0x1FFE,0x0FDE,0x001E,0x00
1E,0x001E,0x001E,0x001E, // Ascii = [q]
```

```
0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x1F7F,0x1FFF,0x1FE7,0x1FC7,0x1F87,0x
1F00,0x1F00,0x1F00,0x1F00,0x1F00,0x1F00,0x1F00,0x1F00,0x1F00,0x1F00,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [r]

0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x07FC,0x1FFE,0x1E0E,0x3E00,0x3E00,0x
3F00,0x1FE0,0x07FC,0x00FE,0x003E,0x001E,0x001E,0x3C3E,0x3FFC,0x1FF0,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [s]

0x0000,0x0000,0x0000,0x0780,0x0780,0x0780,0x7FFF,0x7FFF,0x0780,0x0780,0x0780,0x
0780,0x0780,0x0780,0x0780,0x0780,0x0780,0x0780,0x07C0,0x03FF,0x01FF,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [t]

0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x3C1E,0x3C1E,0x3C1E,0x3C1E,0x3C1E,0x
3C1E,0x3C1E,0x3C1E,0x3C1E,0x3C1E,0x3C3E,0x3C7E,0x3EFE,0x1FFE,0x0FDE,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [u]

0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0xF007,0x780F,0x780F,0x3C1E,0x3C1E,0x
3E1E,0x1E3C,0x1E3C,0x0F78,0x0F78,0x0FF0,0x07F0,0x07F0,0x03E0,0x03E0,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [v]

0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0xF003,0xF1E3,0xF3E3,0xF3E7,0xF3F7,0x
F3F7,0x7FF7,0x7F77,0x7F7F,0x7F7F,0x7F7F,0x3E3E,0x3E3E,0x3E3E,0x3E3E,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [w]

0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x7C0F,0x3E1E,0x3E3C,0x1F3C,0x0FF8,0x
07F0,0x07F0,0x03E0,0x07F0,0x07F8,0x0FF8,0x1E7C,0x3E3E,0x3C1F,0x781F,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [x]

0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0xF807,0x780F,0x7C0F,0x3C1E,0x3C1E,0x
1E3C,0x1E3C,0x1F3C,0x0F78,0x0FF8,0x07F0,0x07F0,0x03E0,0x03E0,0x03C0,0x03C0,0x03
C0,0x0780,0x0F80,0x7F00, // Ascii = [y]

0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x3FFF,0x3FFF,0x001F,0x003E,0x007C,0x
00F8,0x01F0,0x03E0,0x07C0,0x0F80,0x1F00,0x1E00,0x3C00,0x7FFF,0x7FFF,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [z]

0x01FE,0x03E0,0x03C0,0x03C0,0x03C0,0x03C0,0x01E0,0x01E0,0x01E0,0x01C0,0x03C0,0x
3F80,0x3F80,0x03C0,0x01C0,0x01E0,0x01E0,0x01E0,0x03C0,0x03C0,0x03C0,0x03C0,0x03
E0,0x01FE,0x007E,0x0000, // Ascii = [{]

0x01C0,0x01C0,0x01C0,0x01C0,0x01C0,0x01C0,0x01C0,0x01C0,0x01C0,0x01C0,0x01C0,0x
01C0,0x01C0,0x01C0,0x01C0,0x01C0,0x01C0,0x01C0,0x01C0,0x01C0,0x01C0,0x01C0,0x01
C0,0x01C0,0x01C0,0x0000, // Ascii = [|]

0x3FC0,0x03E0,0x01E0,0x01E0,0x01E0,0x01E0,0x01C0,0x03C0,0x03C0,0x01C0,0x01E0,0x
00FE,0x00FE,0x01E0,0x01C0,0x03C0,0x03C0,0x01C0,0x01E0,0x01E0,0x01E0,0x01E0,0x03
E0,0x3FC0,0x3F00,0x0000, // Ascii = [}]

0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x
3F07,0x7FC7,0x73E7,0xF1FF,0xF07E,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x00
00,0x0000,0x0000,0x0000, // Ascii = [~]
};

DisplayFont font_small = {7, 10, font_small_data};
DisplayFont font_medium = {11, 18, font_medium_data};
DisplayFont font_large = {16, 26, font_large_data};

static void send_command(uint8_t cmd) {
    if (i2c_handle != NULL) {
        HAL_I2C_Mem_Write(i2c_handle, DISPLAY_I2C_ADDR, 0x00, 1, &cmd, 1, 10);
    }
}

uint8_t display_init(I2C_HandleTypeDef *hi2c) {
    i2c_handle = hi2c;

    HAL_Delay(100);

    send_command(0xAE);
    send_command(0x20);
    send_command(0x10);
    send_command(0xB0);
    send_command(0xC8);
    send_command(0x00);
    send_command(0x10);
    send_command(0x40);
    send_command(0x81);
    send_command(0xFF);
    send_command(0xA1);
    send_command(0xA6);
    send_command(0xA8);
    send_command(0x3F);
    send_command(0xA4);
    send_command(0xD3);
    send_command(0x00);
    send_command(0xD5);
    send_command(0xF0);
    send_command(0xD9);
    send_command(0x22);
    send_command(0xDA);
    send_command(0x12);
    send_command(0xDB);
    send_command(0x20);
    send_command(0x8D);
    send_command(0x14);
    send_command(0xAF);

    display_fill(COLOR_BLACK);
    display_update();

    display_state.cursor_x = 0;
    display_state.cursor_y = 0;
    display_state.is_inverted = 0;
    display_state.is_initialized = 1;
```

```c
    return 1;
}

void display_fill(DisplayColor color) {
    uint8_t fill_value = (color == COLOR_BLACK) ? 0x00 : 0xFF;

    for (uint32_t i = 0; i < sizeof(display_buffer); i++) {
        display_buffer[i] = fill_value;
    }
}

void display_update(void) {
    if (i2c_handle == NULL) return;

    for (uint8_t page = 0; page < 8; page++) {
        send_command(0xB0 + page);
        send_command(0x00);
        send_command(0x10);

        HAL_I2C_Mem_Write(i2c_handle, DISPLAY_I2C_ADDR, 0x40, 1,
                         &display_buffer[DISPLAY_WIDTH * page],
                         DISPLAY_WIDTH, 25);
    }
}

void display_set_pixel(uint8_t x, uint8_t y, DisplayColor color) {
    if (x >= DISPLAY_WIDTH || y >= DISPLAY_HEIGHT) return;

    if (display_state.is_inverted) {
        color = (color == COLOR_BLACK) ? COLOR_WHITE : COLOR_BLACK;
    }

    uint16_t index = x + (y / 8) * DISPLAY_WIDTH;

    if (color == COLOR_WHITE) {
        display_buffer[index] |= (1 << (y % 8));
    } else {
        display_buffer[index] &= ~(1 << (y % 8));
    }
}

void display_set_cursor(uint8_t x, uint8_t y) {
    display_state.cursor_x = x;
    display_state.cursor_y = y;
}

void display_clear(void) {
    display_fill(COLOR_BLACK);
    display_update();
    display_set_cursor(0, 0);
```

```c
}

char display_write_char(char ch, DisplayFont *font, DisplayColor color) {
    if (display_state.cursor_x + font->width > DISPLAY_WIDTH ||
        display_state.cursor_y + font->height > DISPLAY_HEIGHT) {
        return 0;
    }

    if (ch < 32 || ch > 126) ch = ' ';

    for (uint8_t row = 0; row < font->height; row++) {
        uint16_t glyph_row = font->glyph_data[(ch - 32) * font->height + row];

        for (uint8_t col = 0; col < font->width; col++) {
            if (glyph_row & (1 << (15 - col))) {
                display_set_pixel(display_state.cursor_x + col,
                                display_state.cursor_y + row,
                                color);
            } else {
                display_set_pixel(display_state.cursor_x + col,
                                display_state.cursor_y + row,
                                (color == COLOR_WHITE) ? COLOR_BLACK :
COLOR_WHITE);
            }
        }
    }

    display_state.cursor_x += font->width;
    return ch;
}

char display_write_string(char *str, DisplayFont *font, DisplayColor color) {
    while (*str) {
        if (display_write_char(*str, font, color) != *str) {
            return *str;
        }
        str++;
    }
    return *str;
}
```

display.h

```c
#ifndef DISPLAY_H
#define DISPLAY_H

#include "stm32f4xx_hal.h"

#define DISPLAY_WIDTH  128
#define DISPLAY_HEIGHT 64

#define DISPLAY_I2C_ADDR 0x78

typedef enum {
    COLOR_BLACK = 0x00,
    COLOR_WHITE = 0x01
} DisplayColor;

typedef struct {
    uint8_t width;
    uint8_t height;
    const uint16_t *glyph_data;
} DisplayFont;

typedef struct {
    uint16_t cursor_x;
    uint16_t cursor_y;
    uint8_t is_inverted;
    uint8_t is_initialized;
} DisplayState;

extern DisplayFont font_small;
extern DisplayFont font_medium;
extern DisplayFont font_large;

uint8_t display_init(I2C_HandleTypeDef *hi2c);
void display_fill(DisplayColor color);
void display_update(void);
void display_set_pixel(uint8_t x, uint8_t y, DisplayColor color);
void display_set_cursor(uint8_t x, uint8_t y);
char display_write_char(char ch, DisplayFont *font, DisplayColor color);
char display_write_string(char *str, DisplayFont *font, DisplayColor color);
void display_clear(void);

#endif
```

main.c

```
/* USER CODE BEGIN Header */
/**

******************************************************************************
  * @file           : main.c
  * @brief          : Main program body

******************************************************************************
  * @attention
  *
  * Copyright (c) 2022 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"

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

#include "display.h"
#include <stdio.h>
#include <stdarg.h>
#include <usart.h>
#include <string.h>
#include <ctype.h>
#include <ctype.h>
#include <stdbool.h>


/* USER CODE END Includes */

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

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

/* Private define
------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define UART_TIMEOUT 10
#define BUF_SIZE 256
#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 (200)
#define DISPLAY_UPDATE_CYCLES 64

/* USER CODE END PD */

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

/* USER CODE END PM */

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

/* USER CODE BEGIN PV */
typedef enum {
    OPERATION_NONE,
    OPERATION_ADD,
    OPERATION_SUBTRACT,
    OPERATION_MULTIPLY,
    OPERATION_DIVIDE
} CalculatorOperation;

typedef struct {
    int32_t first_operand;
    int32_t second_operand;
    CalculatorOperation operation;
    uint8_t is_first_operand_active;
    uint8_t is_second_operand_active;
    uint8_t is_result_shown;
    uint8_t operation_cycle_count;
} CalculatorState;

CalculatorState calc_state = {0};

uint32_t last_pressing_time = 0;
int last_pressed_btn_index = -1;
uint8_t display_update_counter = 0;
```

```c
uint8_t display_busy = 0;

char calc_layout[] = {
    '3', '2', '1',
    '6', '5', '4',
    '9', '8', '7',
    'x', 'e', '0'
};

CalculatorOperation operation_cycle[4] = {
    OPERATION_ADD,
    OPERATION_SUBTRACT,
    OPERATION_MULTIPLY,
    OPERATION_DIVIDE
};
uint8_t current_op_index = 0;

/* USER CODE END PV */

/* Private function prototypes
-----------------------------------------------*/
void SystemClock_Config(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

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

void calculator_reset(void) {
    calc_state.first_operand = 0;
    calc_state.second_operand = 0;
    calc_state.operation = OPERATION_NONE;
    calc_state.is_first_operand_active = 1;
    calc_state.is_second_operand_active = 0;
    calc_state.is_result_shown = 0;
    calc_state.operation_cycle_count = 0;
    current_op_index = 0;
}

void calculator_add_digit(uint8_t digit) {
    if (calc_state.is_result_shown) {
        calculator_reset();
    }

    if (calc_state.is_first_operand_active) {
        if (calc_state.first_operand > 9999) {
            return;
        }
        calc_state.first_operand = calc_state.first_operand * 10 + digit;
    } else {
        if (calc_state.second_operand > 9999) {
            return;
        }
        calc_state.second_operand = calc_state.second_operand * 10 + digit;
        calc_state.is_second_operand_active = 1;
    }
}

void calculator_set_operation(CalculatorOperation op) {
    if (calc_state.is_first_operand_active && calc_state.first_operand != 0) {
        calc_state.operation = op;
        calc_state.is_first_operand_active = 0;
        calc_state.operation_cycle_count = 0;
    }
}

void calculator_cycle_operation(void) {
    //if (calc_state.is_first_operand_active && calc_state.first_operand != 0)
{
        current_op_index = (current_op_index + 1) % 4;
        calc_state.operation = operation_cycle[current_op_index];
        calc_state.is_first_operand_active = 0;
        calc_state.operation_cycle_count++;

        if (calc_state.operation_cycle_count > 4) {
            calc_state.operation_cycle_count = 1;
        }
    //}
}

int32_t calculator_calculate(void) {
    int32_t result = 0;

    switch (calc_state.operation) {
        case OPERATION_ADD:
            result = calc_state.first_operand + calc_state.second_operand;
            break;
        case OPERATION_SUBTRACT:
            result = calc_state.first_operand - calc_state.second_operand;
            break;
        case OPERATION_MULTIPLY:
            result = calc_state.first_operand * calc_state.second_operand;
            break;
        case OPERATION_DIVIDE:
            if (calc_state.second_operand != 0) {
                result = calc_state.first_operand / calc_state.second_operand;
            } else {
                result = 0;
            }
            break;
        default:
            result = calc_state.first_operand;
            break;
    }

    calc_state.is_result_shown = 1;
    return result;
}

char get_operation_symbol(void) {
    switch (calc_state.operation) {
        case OPERATION_ADD: return '+';
        case OPERATION_SUBTRACT: return '-';
        case OPERATION_MULTIPLY: return '*';
        case OPERATION_DIVIDE: return '/';
        default: return ' ';
    }
}

void update_display(void) {
    char buffer[32];

    display_clear();

    bool first_too_long = (calc_state.first_operand > 99999);
    bool second_too_long = (calc_state.second_operand > 99999);

    display_set_cursor(0, 0);
    if (calc_state.first_operand != 0 || calc_state.is_first_operand_active) {
        sprintf(buffer, "%ld", calc_state.first_operand);
        display_write_string(buffer, &font_medium, COLOR_WHITE);

        if (calc_state.operation != OPERATION_NONE) {
            char op[2] = {get_operation_symbol(), '\0'};
            display_write_string(op, &font_medium, COLOR_WHITE);
        }

        if (first_too_long) {
            display_set_cursor(64, 0);
            display_write_string(">", &font_medium, COLOR_WHITE);
            display_set_cursor(70, 0);
            display_write_string("5", &font_medium, COLOR_WHITE);

            display_set_cursor(64, 0);
            display_write_string(">", &font_medium, COLOR_BLACK);
            display_set_cursor(70, 0);
            display_write_string("5", &font_medium, COLOR_BLACK);
        }
    }
    display_set_cursor(0, 20);
    if (calc_state.is_second_operand_active) {
        sprintf(buffer, "%ld=", calc_state.second_operand);
        display_write_string(buffer, &font_medium, COLOR_WHITE);

        if (second_too_long) {
            display_set_cursor(64, 20);
            display_write_string(">", &font_medium, COLOR_WHITE);
            display_set_cursor(70, 20);
            display_write_string("5", &font_medium, COLOR_WHITE);

            display_set_cursor(64, 20);
            display_write_string(">", &font_medium, COLOR_BLACK);
            display_set_cursor(70, 20);
            display_write_string("5", &font_medium, COLOR_BLACK);
        }
    } else if (calc_state.operation != OPERATION_NONE &&
!calc_state.is_first_operand_active) {
        display_write_string("=", &font_medium, COLOR_WHITE);
    }

    display_set_cursor(0, 40);
    if (calc_state.is_result_shown) {
        int32_t result = calculator_calculate();
        sprintf(buffer, "%ld", result);
        display_write_string(buffer, &font_medium, COLOR_WHITE);

        if (result > 99999 || result < -9999) {
            display_set_cursor(64, 40);
            display_write_string(">", &font_medium, COLOR_WHITE);
            display_set_cursor(70, 40);
            display_write_string("5", &font_medium, COLOR_WHITE);

            display_set_cursor(64, 40);
            display_write_string(">", &font_medium, COLOR_BLACK);
            display_set_cursor(70, 40);
            display_write_string("5", &font_medium, COLOR_BLACK);
        }
    }

    display_update();
}

char key2char(int key_index) {
    if (key_index < 0 || key_index >= 12) {
        return '!';
    }
    return calc_layout[key_index];
}
```

```
void process_calculator_input(char input_char) {
    char uart_msg[64];

    sprintf(uart_msg, "\r\nКлавиша: %c\r\n", input_char);
    HAL_UART_Transmit(&huart6, (uint8_t*)uart_msg, strlen(uart_msg), 100);

    if (input_char >= '0' && input_char <= '9') {
        uint8_t digit = input_char - '0';
        calculator_add_digit(digit);

        sprintf(uart_msg, "Введена цифра: %d\r\n", digit);
        HAL_UART_Transmit(&huart6, (uint8_t*)uart_msg, strlen(uart_msg), 100);
    }
    else if (input_char == 'C' || input_char == 'c') {
        calculator_reset();
        HAL_UART_Transmit(&huart6, (uint8_t*)"\r\nСброс калькулятора\r\n", 22,
100);
    }
    else if (input_char == '=') {
        if (calc_state.operation != OPERATION_NONE &&
calc_state.is_second_operand_active) {
            int32_t result = calculator_calculate();
            sprintf(uart_msg, "\r\nРезультат: %ld\r\n", result);
            HAL_UART_Transmit(&huart6, (uint8_t*)uart_msg, strlen(uart_msg),
100);
        } else {
            HAL_UART_Transmit(&huart6, (uint8_t*)"\r\nНевозможно
вычислить\r\n", 23, 100);
        }
    }
    else if (input_char == '+') {

    }

    update_display();
}

void handle_button_press(int btn_index) {
    char received_char = key2char(btn_index);

    char msg[16];
    sprintf(msg, "\r\n%c", received_char);
    HAL_UART_Transmit(&huart6, (uint8_t*)msg, strlen(msg), 5);

    if (received_char >= '0' && received_char <= '9') {
        uint8_t digit = received_char - '0';

        if (calc_state.is_first_operand_active) {
            if (calc_state.first_operand > 9999) {
                HAL_UART_Transmit(&huart6, (uint8_t*)"\r\nСлишком длинное число
(>5 цифр)!\r\n", 40, 100);
                return;
            }
        } else {
            if (calc_state.second_operand > 9999) {
                HAL_UART_Transmit(&huart6, (uint8_t*)"\r\nСлишком длинное число
(>5 цифр)!\r\n", 40, 100);
                return;
            }
        }

        calculator_add_digit(digit);
    }
    else if (received_char == 'x') {
        calculator_reset();
        HAL_UART_Transmit(&huart6, (uint8_t*)"\r\nCLR", 4, 5);
    }
    else if (received_char == 'e') {
        calculator_cycle_operation();
        char op_symbol = get_operation_symbol();
        char op_msg[4] = "\r\n \0";
        op_msg[2] = op_symbol;
        HAL_UART_Transmit(&huart6, (uint8_t*)op_msg, 3, 5);
    }

    update_display();
}

void calculator_enter(void) {
    if (calc_state.operation != OPERATION_NONE &&
calc_state.is_second_operand_active) {
        int32_t result = calculator_calculate();
        char uart_msg[64];
        sprintf(uart_msg, "\r\nРезультат: %ld\r\n", result);
        HAL_UART_Transmit(&huart6, (uint8_t*)uart_msg, strlen(uart_msg), 100);
        update_display();
    } else {
        HAL_UART_Transmit(&huart6, (uint8_t*)"\r\nНевозможно вычислить\r\n",
23, 100);
    }
}

static void set_green_led(bool on) {
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_13, on ? GPIO_PIN_SET : GPIO_PIN_RESET);
}

static void set_yellow_led(bool on) {
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_14, on ? GPIO_PIN_SET : GPIO_PIN_RESET);
}

static void set_red_led(bool on) {
    HAL_GPIO_WritePin(GPIOD, GPIO_PIN_15, on ? GPIO_PIN_SET : GPIO_PIN_RESET);
}

void blink_leds(int count) {
    for (int i = 0; i < count; i++) {
        set_green_led(true);
        set_yellow_led(true);
        set_red_led(true);
        HAL_Delay(100);
        set_green_led(false);
        set_yellow_led(false);
        set_red_led(false);
        if (i < count - 1) HAL_Delay(100);
    }
}

bool is_mode_btn_pressed(void) {
    return HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_15) == GPIO_PIN_RESET;
}

int get_pressed_btn_index(void) {
    const uint32_t t = HAL_GetTick();

    if (t - last_pressing_time < KB_KEY_DEBOUNCE_TIME) {
        return -1;
    }

    int index = -1;
    uint8_t reg_buffer = 0xFF;
    uint8_t tmp = 0;
    int detected_keys = 0;
    int row_values[4] = {0};
    int col_values[3] = {0};

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

    for (int row = 0; row < 4; row++) {
        uint8_t config_value = ~((uint8_t)(1 << row));
        HAL_I2C_Mem_Write(&hi2c1, KB_I2C_WRITE_ADDRESS, KB_CONFIG_REG, 1,
&config_value, 1, KB_KEY_DEBOUNCE_TIME);

        HAL_Delay(2);

        HAL_I2C_Mem_Read(&hi2c1, KB_I2C_READ_ADDRESS, KB_INPUT_REG, 1,
&reg_buffer, 1, KB_KEY_DEBOUNCE_TIME);

        uint8_t column_state = (reg_buffer >> 4) & 0x07;

        if ((column_state & 0x04) == 0) {
            if (detected_keys == 0) index = row * 3 + 0;
            detected_keys++;
            row_values[row]++;
            col_values[0]++;
        }
        if ((column_state & 0x02) == 0) {
            if (detected_keys == 0) index = row * 3 + 1;
            detected_keys++;
            row_values[row]++;
            col_values[1]++;
        }
        if ((column_state & 0x01) == 0) {
            if (detected_keys == 0) index = row * 3 + 2;
            detected_keys++;
            row_values[row]++;
            col_values[2]++;
        }
    }

    if (detected_keys > 2) {
        return -1;
    }

    int rows_pressed = 0;
    int cols_pressed = 0;
    for (int i = 0; i < 4; i++) {
        if (row_values[i] > 0) rows_pressed++;
    }
    for (int i = 0; i < 3; i++) {
        if (col_values[i] > 0) cols_pressed++;
    }

    if (rows_pressed != 1 || cols_pressed != 1) {
        return -1;
    }

    if (detected_keys == 1 && index != -1) {
        last_pressed_btn_index = index;
        last_pressing_time = t;
        return index;
    }

    if (detected_keys == 0) {
        last_pressed_btn_index = -1;
    }

    return -1;
}

void check_and_update_display(void) {
    if (display_busy) {
        return;
    }

    display_update_counter++;

    if (display_update_counter >= DISPLAY_UPDATE_CYCLES) {
        display_update_counter = 0;
        update_display();
    }
}


/* 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_USART6_UART_Init();
  MX_TIM1_Init();
  MX_TIM6_Init();
  MX_I2C1_Init();
  /* USER CODE BEGIN 2 */
  HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
      HAL_TIM_Base_Start_IT(&htim6);
      uint8_t init_config = 0xFF;
        HAL_I2C_Mem_Write(&hi2c1, KB_I2C_WRITE_ADDRESS, KB_CONFIG_REG, 1,
&init_config, 1, 100);

        set_green_led(false);
        set_yellow_led(false);
        set_red_led(false);
        bool display_initialized = false;
          if (display_init(&hi2c1)) {
              display_initialized = true;
              display_clear();

              display_set_cursor(0, 0);
              display_write_string("Calculator", &font_medium, COLOR_WHITE);
              display_set_cursor(0, 20);
              display_write_string("SDK-1.1M Ready", &font_small, COLOR_WHITE);
              display_update();
              HAL_Delay(2000);

              calculator_reset();
              update_display();
          } else {
              char error_msg[] = "\r\nDisplay init failed!\r\n";
              HAL_UART_Transmit(&huart6, (uint8_t*)error_msg,
strlen(error_msg), 100);
          }
          char welcome_message[] = "\r\n"
                "Калькулятор с OLED дисплеем\r\n"
                "Клавиатура:\r\n"
                "  0-9 - цифры\r\n"
                "  x   - сброс (C)\r\n"
                "  e   - переключение операции (+, -, *, /)\r\n"
                "  =   - вычисление (используется автоматически при вводе
второго числа)\r\n"
                "========================================\r\n";
            HAL_UART_Transmit(&huart6, (uint8_t*)welcome_message,
strlen(welcome_message), 100);

            blink_leds(2);

            uint32_t last_mode_change = 0;
            bool mode_btn_pressed = false;
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
            while (1)
              {
                  bool current_btn_state = is_mode_btn_pressed();

                  if (current_btn_state && !mode_btn_pressed) {
                      mode_btn_pressed = true;
                      last_mode_change = HAL_GetTick();
                  }
                  else if (!current_btn_state && mode_btn_pressed) {
                      mode_btn_pressed = false;
                      uint32_t press_duration = HAL_GetTick() -
last_mode_change;

                      if (press_duration > 50 && press_duration < 500) {
                          calculator_enter();
                      }
                      else if (press_duration >= 1000) {
                          calculator_reset();
                          update_display();
                          HAL_UART_Transmit(&huart6, (uint8_t*)"\r\nCalculator
Reset\r\n", 20, 100);
                      }
                  }

                  int btn_index = get_pressed_btn_index();

                  if (btn_index != -1) {
                      handle_button_press(btn_index);

                      if (display_initialized) {
                          update_display();
                      }
                  }

                  if (display_initialized) {
                      check_and_update_display();
                  }

                  HAL_Delay(10);
    /* 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_SCALE1);

  /** 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 = 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 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_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();
  }
}

/* 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 работы был создан калькулятор на SDK-1.1M с OLED-дисплеем и клавиатурой, работающими через один интерфейс I^{2}C. UART теперь используется только для информационного вывода. Организовано бесконфликтное взаимодействие устройств на шине за счет временного разделения доступа. Реализован механизм периодического обновления дисплея через счетчик прерываний, что снижает нагрузку на шину. Калькулятор поддерживает основные операции с защитой от ошибок, циклический выбор операций и автоматическое вычисление при вводе второго числа. Отработана система обработки нажатий клавиш с устранением дребезга.