mode0_demo.c

```c
#include "pico/stdlib.h"
#include "mode0/mode0.h"


int main() {
    mode0_init();

    mode0_set_cursor(0, 0);
    mode0_color_t fg = MODE0_WHITE;
    mode0_color_t bg = MODE0_BLACK;

    while (1) {
        mode0_print("Retro Computer (c) 2021, Shawn Hyam\n");
        sleep_ms(500);
        fg = (fg+1) % 16;
        if (fg == 0) {
            bg = (bg+1) % 16;
            mode0_set_background(bg);
        }
        mode0_set_foreground(fg);

    }
}
```

mode0.h

```c
#ifndef _TEXT_MODE_H
#define _TEXT_MODE_H

// ARNE-16 palette converted to RGB565 -- https://lospec.com/palette-list/arne-16
typedef enum {
    MODE0_BLACK,
    MODE0_BROWN,
    MODE0_RED,
    MODE0_BLUSH,
    MODE0_GRAY,
    MODE0_DESERT,
    MODE0_ORANGE,
    MODE0_YELLOW,
    MODE0_WHITE,
    MODE0_MIDNIGHT,
    MODE0_DARK_SLATE_GRAY,
    MODE0_GREEN,
    MODE0_YELLOW_GREEN,
    MODE0_BLUE,
    MODE0_PICTON_BLUE,
    MODE0_PALE_BLUE
} mode0_color_t;

void mode0_init();
void mode0_clear(mode0_color_t color);
void mode0_draw_screen();
void mode0_draw_region(uint8_t x, uint8_t y, uint8_t width, uint8_t height);
void mode0_scroll_vertical(int8_t amount);
void mode0_set_foreground(mode0_color_t color);
void mode0_set_background(mode0_color_t color);
void mode0_set_cursor(uint8_t x, uint8_t y);
uint8_t mode0_get_cursor_x();
uint8_t mode0_get_cursor_y();
void mode0_print(const char *s);
void mode0_write(const char *s, int len);
void mode0_putc(char c);
void mode0_show_cursor();
void mode0_hide_cursor();

// Won't redraw until the matching _end is invoked.
void mode0_begin();
void mode0_end();

#endif
```

mode0.c

```c
#include "pico/stdlib.h"
#include <string.h>
#include "hardware/spi.h"
#include "ili9341/ili9341.h"
#include "mode0/mode0.h"

/* Character graphics mode */

// Characters are 8x12 -- characters start at (x:1,y:1) and are 5x7 in size, so
// it is possible to not display the full area. This display mode actually treats
// them as 6x10, starting at (x:1,y:0)
static const uint8_t font_data[95][12] = {
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x10, 0x10, 0x10, 0x10, 0x10, 0x00, 0x10, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x28, 0x28, 0x28, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x00, 0x28, 0x7C, 0x28, 0x7C, 0x28, 0x00, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x10, 0x3C, 0x40, 0x38, 0x04, 0x78, 0x10, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x60, 0x64, 0x08, 0x10, 0x20, 0x4C, 0x0C, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x20, 0x50, 0x50, 0x20, 0x54, 0x48, 0x34, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x10, 0x10, 0x10, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x08, 0x10, 0x20, 0x20, 0x20, 0x10, 0x08, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x20, 0x10, 0x08, 0x08, 0x08, 0x10, 0x20, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x00, 0x10, 0x54, 0x38, 0x38, 0x54, 0x10, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x00, 0x10, 0x10, 0x7C, 0x10, 0x10, 0x00, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x18, 0x10, 0x20, 0x00, 0x00 },
    { 0x00, 0x00, 0x00, 0x00, 0x7C, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x18, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x00, 0x04, 0x08, 0x10, 0x20, 0x40, 0x00, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x38, 0x44, 0x4C, 0x54, 0x64, 0x44, 0x38, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x10, 0x30, 0x10, 0x10, 0x10, 0x10, 0x38, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x38, 0x44, 0x04, 0x38, 0x40, 0x40, 0x7C, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x38, 0x44, 0x04, 0x18, 0x04, 0x44, 0x38, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x08, 0x18, 0x28, 0x48, 0x7C, 0x08, 0x08, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x7C, 0x40, 0x78, 0x04, 0x04, 0x44, 0x38, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x18, 0x20, 0x40, 0x78, 0x44, 0x44, 0x38, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x7C, 0x04, 0x08, 0x10, 0x20, 0x40, 0x40, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x38, 0x44, 0x44, 0x38, 0x44, 0x44, 0x38, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x38, 0x44, 0x44, 0x3C, 0x04, 0x08, 0x30, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x00, 0x00, 0x30, 0x30, 0x00, 0x30, 0x30, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x00, 0x00, 0x30, 0x30, 0x00, 0x30, 0x30, 0x10, 0x20, 0x00, 0x00 },
    { 0x00, 0x08, 0x10, 0x20, 0x40, 0x20, 0x10, 0x08, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x00, 0x00, 0x7C, 0x00, 0x7C, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x40, 0x20, 0x10, 0x08, 0x10, 0x20, 0x40, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x38, 0x44, 0x04, 0x08, 0x10, 0x00, 0x10, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x38, 0x44, 0x04, 0x34, 0x4C, 0x44, 0x38, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x10, 0x28, 0x44, 0x44, 0x7C, 0x44, 0x44, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x78, 0x44, 0x44, 0x78, 0x44, 0x44, 0x78, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x38, 0x44, 0x40, 0x40, 0x40, 0x44, 0x38, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x78, 0x44, 0x44, 0x44, 0x44, 0x44, 0x78, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x7C, 0x40, 0x40, 0x70, 0x40, 0x40, 0x7C, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x7C, 0x40, 0x40, 0x70, 0x40, 0x40, 0x40, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x38, 0x44, 0x40, 0x4C, 0x44, 0x44, 0x38, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x44, 0x44, 0x44, 0x7C, 0x44, 0x44, 0x44, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x38, 0x10, 0x10, 0x10, 0x10, 0x10, 0x38, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x04, 0x04, 0x04, 0x04, 0x44, 0x44, 0x38, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x44, 0x48, 0x50, 0x60, 0x50, 0x48, 0x44, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x40, 0x40, 0x40, 0x40, 0x40, 0x40, 0x7C, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x44, 0x6C, 0x54, 0x54, 0x44, 0x44, 0x44, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x44, 0x44, 0x64, 0x54, 0x4C, 0x44, 0x44, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x38, 0x44, 0x44, 0x44, 0x44, 0x44, 0x38, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x78, 0x44, 0x44, 0x78, 0x40, 0x40, 0x40, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x38, 0x44, 0x44, 0x44, 0x54, 0x48, 0x34, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x78, 0x44, 0x44, 0x78, 0x50, 0x48, 0x44, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x38, 0x44, 0x40, 0x38, 0x04, 0x44, 0x38, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x7C, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x44, 0x44, 0x44, 0x44, 0x44, 0x44, 0x38, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x44, 0x44, 0x44, 0x28, 0x28, 0x10, 0x10, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x44, 0x44, 0x44, 0x54, 0x54, 0x6C, 0x44, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x44, 0x44, 0x28, 0x10, 0x28, 0x44, 0x44, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x44, 0x44, 0x28, 0x10, 0x10, 0x10, 0x10, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x7C, 0x04, 0x08, 0x10, 0x20, 0x40, 0x7C, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x38, 0x20, 0x20, 0x20, 0x20, 0x20, 0x38, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x00, 0x40, 0x20, 0x10, 0x08, 0x04, 0x00, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x38, 0x08, 0x08, 0x08, 0x08, 0x08, 0x38, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x10, 0x28, 0x44, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x7C, 0x00, 0x00, 0x00 },
    { 0x00, 0x20, 0x10, 0x08, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x00, 0x00, 0x38, 0x04, 0x3C, 0x44, 0x3C, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x40, 0x40, 0x58, 0x64, 0x44, 0x44, 0x78, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x00, 0x00, 0x38, 0x44, 0x40, 0x44, 0x38, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x04, 0x04, 0x34, 0x4C, 0x44, 0x44, 0x3C, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x00, 0x00, 0x38, 0x44, 0x78, 0x40, 0x38, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x10, 0x28, 0x20, 0x70, 0x20, 0x20, 0x20, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x00, 0x00, 0x3C, 0x44, 0x44, 0x4C, 0x34, 0x04, 0x38, 0x00, 0x00 },
    { 0x00, 0x40, 0x40, 0x58, 0x64, 0x44, 0x44, 0x44, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x10, 0x00, 0x30, 0x10, 0x10, 0x10, 0x38, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x00, 0x08, 0x00, 0x08, 0x08, 0x08, 0x08, 0x48, 0x30, 0x00, 0x00 },
    { 0x00, 0x40, 0x40, 0x48, 0x50, 0x60, 0x50, 0x48, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x30, 0x10, 0x10, 0x10, 0x10, 0x10, 0x38, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x00, 0x00, 0x68, 0x54, 0x54, 0x54, 0x54, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x00, 0x00, 0x58, 0x64, 0x44, 0x44, 0x44, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x00, 0x00, 0x38, 0x44, 0x44, 0x44, 0x38, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x00, 0x00, 0x38, 0x44, 0x44, 0x64, 0x58, 0x40, 0x40, 0x00, 0x00 },
    { 0x00, 0x00, 0x00, 0x38, 0x44, 0x44, 0x4C, 0x34, 0x04, 0x04, 0x00, 0x00 },
    { 0x00, 0x00, 0x00, 0x58, 0x64, 0x40, 0x40, 0x40, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x00, 0x00, 0x3C, 0x40, 0x38, 0x04, 0x78, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x20, 0x70, 0x20, 0x20, 0x20, 0x28, 0x10, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x00, 0x00, 0x44, 0x44, 0x44, 0x4C, 0x34, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x00, 0x00, 0x44, 0x44, 0x44, 0x28, 0x10, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x00, 0x00, 0x44, 0x44, 0x54, 0x54, 0x28, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x00, 0x00, 0x44, 0x28, 0x10, 0x28, 0x44, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x00, 0x00, 0x44, 0x44, 0x44, 0x44, 0x3C, 0x04, 0x38, 0x00, 0x00 },
    { 0x00, 0x00, 0x00, 0x7C, 0x08, 0x10, 0x20, 0x7C, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x20, 0x10, 0x10, 0x08, 0x10, 0x10, 0x20, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x10, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x08, 0x10, 0x10, 0x20, 0x10, 0x10, 0x08, 0x00, 0x00, 0x00, 0x00 },
    { 0x00, 0x28, 0x50, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 }
};


#define TEXT_HEIGHT 24
#define TEXT_WIDTH 53

#define SWAP_BYTES(color) ((uint16_t)(color>>8) | (uint16_t)(color<<8))

static mode0_color_t screen_bg_color = MODE0_BLACK;
static mode0_color_t screen_fg_color = MODE0_WHITE;  // TODO need to store a color per cell
static int cursor_x = 0;
static int cursor_y = 0;
static uint8_t screen[TEXT_HEIGHT * TEXT_WIDTH] = { 0 };
static uint8_t colors[TEXT_HEIGHT * TEXT_WIDTH] = { 0 };
static uint8_t show_cursor = 0;

static int depth = 0;
static uint16_t palette[16] = {
    SWAP_BYTES(0x0000),
    SWAP_BYTES(0x49E5),
    SWAP_BYTES(0xB926),
    SWAP_BYTES(0xE371),
    SWAP_BYTES(0x9CF3),
    SWAP_BYTES(0xA324),
    SWAP_BYTES(0xEC46),
    SWAP_BYTES(0xF70D),
    SWAP_BYTES(0xffff),
    SWAP_BYTES(0x1926),
    SWAP_BYTES(0x2A49),
    SWAP_BYTES(0x4443),
    SWAP_BYTES(0xA664),
    SWAP_BYTES(0x02B0),
    SWAP_BYTES(0x351E),
    SWAP_BYTES(0xB6FD)
};

void mode0_clear(mode0_color_t color) {
    mode0_begin();
    int size = TEXT_WIDTH*TEXT_HEIGHT;
    memset(screen, 0, size);
    memset(colors, color, size);
    mode0_set_cursor(0, 0);
    mode0_end();
}

void mode0_set_foreground(mode0_color_t color) {
    mode0_begin();
    screen_fg_color = color;
    mode0_end();
}

void mode0_set_background(mode0_color_t color) {
    mode0_begin();
    screen_bg_color = color;
    mode0_end();
}

void mode0_set_cursor(uint8_t x, uint8_t y) {
    cursor_x = x;
    cursor_y = y;
}

void mode0_show_cursor() {
    mode0_begin();
    show_cursor = 1;
    mode0_end();
}

void mode0_hide_cursor() {
    mode0_begin();
    show_cursor = 0;
    mode0_end();
}

uint8_t mode0_get_cursor_x() {
    return cursor_x;
}

uint8_t mode0_get_cursor_y() {
    return cursor_y;
}

void mode0_putc(char c) {
    mode0_begin();

    if (cursor_y >= TEXT_HEIGHT) {
        mode0_scroll_vertical(cursor_y-TEXT_HEIGHT+1);
        cursor_y = TEXT_HEIGHT-1;
    }

    int idx = cursor_y*TEXT_WIDTH + cursor_x;
    if (c == '\n') {
        // fill the rest of the line with empty content + the current bg color
        memset(screen+idx, 0, TEXT_WIDTH-cursor_x);
        memset(colors+idx, screen_bg_color, TEXT_WIDTH-cursor_x);
        cursor_y++;
        cursor_x = 0;
    } else if (c == '\r') {
        //cursor_x = 0;
    } else if (c>=32 && c<=127) {
        screen[idx] = c-32;
        colors[idx] = ((screen_fg_color & 0xf) << 4) | (screen_bg_color & 0xf);

        cursor_x++;
        if (cursor_x >= TEXT_WIDTH) {
            cursor_x = 0;
            cursor_y++;
        }
    }

    mode0_end();
}

void mode0_print(const char *str) {
    mode0_begin();
    char c;
    while (c = *str++) {
        mode0_putc(c);
    }
    mode0_end();
}

void mode0_write(const char *str, int len) {
    mode0_begin();
    for (int i=0; i<len; i++) {
        mode0_putc(*str++);
    }
    mode0_end();
}

inline void mode0_begin() {
    depth++;
}
```

```
inline void mode0_end() {
    if (--depth == 0) {
        mode0_draw_screen();
    }
}

void mode0_draw_region(uint8_t x, uint8_t y, uint8_t width, uint8_t height) {
    // TODO
    mode0_draw_screen();
}

void mode0_draw_screen() {
    // assert depth == 0?
    depth = 0;

    // setup to draw the whole screen

    // column address set
    ili9341_set_command(ILI9341_CASET);
    ili9341_command_param(0x00);
    ili9341_command_param(0x00);  // start column
    ili9341_command_param(0x00);
    ili9341_command_param(0xef);  // end column -> 239

    // page address set
    ili9341_set_command(ILI9341_PASET);
    ili9341_command_param(0x00);
    ili9341_command_param(0x00);  // start page
    ili9341_command_param(0x01);
    ili9341_command_param(0x3f);  // end page -> 319

    // start writing
    ili9341_set_command(ILI9341_RAMWR);

    uint16_t buffer[6*240];  // 'amount' pixels wide, 240 pixels tall

    int screen_idx = 0;
    for (int x=0; x<TEXT_WIDTH; x++) {
        // create one column of screen information

        uint16_t *buffer_idx = buffer;

        for (int bit=0; bit<6; bit++) {
            uint8_t mask = 64>>bit;
            for (int y=TEXT_HEIGHT-1; y>=0; y--) {
                uint8_t character = screen[y*53+x];
                uint16_t fg_color = palette[colors[y*53+x] >> 4];
                uint16_t bg_color = palette[colors[y*53+x] & 0xf];

                if (show_cursor && (cursor_x == x) && (cursor_y == y)) {
                    bg_color = MODE0_GREEN;
                }

                const uint8_t* pixel_data = font_data[character];

                // draw the character into the buffer
                for (int j=10; j>=1; j--) {
                    *buffer_idx++ = (pixel_data[j] & mask) ? fg_color : bg_color;
                }
            }
        }

        // now send the slice
        ili9341_write_data(buffer, 6*240*2);
    }

    uint16_t extra_buffer[2*240] = { 0 };
    ili9341_write_data(extra_buffer, 2*240*2);

}

void mode0_scroll_vertical(int8_t amount) {
    mode0_begin();


    if (amount > 0) {
        int size1 = TEXT_WIDTH*amount;
        int size2 = TEXT_WIDTH*TEXT_HEIGHT - size1;

        memmove(screen, screen+size1, size2);
        memmove(colors, colors+size1, size2);
        memset(screen+size2, 0, size1);
        memset(colors+size2, screen_bg_color, size1);
    } else if (amount < 0) {
        amount = -amount;
        int size1 = TEXT_WIDTH*amount;
        int size2 = TEXT_WIDTH*TEXT_HEIGHT - size1;

        memmove(screen+size1, screen, size2);
        memmove(colors+size1, colors, size2);
        memset(screen, 0, size1);
        memset(colors, screen_bg_color, size1);
    }

    mode0_end();
}

void mode0_init() {
    stdio_init_all();
    ili9341_init();
}
```

ili9341.h

```
#ifndef _ILI9341_H
#define _ILI9341_H

#include <stdint.h>
#include "pico/stdlib.h"
#include "hardware/spi.h"

/*
#define SPI_PORT spi0
#define PIN_MISO 4
#define PIN_CS   5
#define PIN_SCK  6
#define PIN_MOSI 7
#define PIN_RESET 8
#define PIN_DC 9  // data/command
*/

typedef struct {
    spi_inst_t *port;
    uint pin_miso;
    uint pin_cs;
    uint pin_sck;
    uint pin_mosi;
    uint pin_reset;
    uint pin_dc;
} ili9341_config_t;

extern ili9341_config_t ili9341_config;

#define ILI9341_TFTWIDTH 240  ///< ILI9341 max TFT width
#define ILI9341_TFTHEIGHT 320 ///< ILI9341 max TFT height

#define ILI9341_NOP 0x00     ///< No-op register
#define ILI9341_SWRESET 0x01 ///< Software reset register
#define ILI9341_RDDID 0x04   ///< Read display identification information
#define ILI9341_RDDST 0x09   ///< Read Display Status

#define ILI9341_SLPIN 0x10  ///< Enter Sleep Mode
#define ILI9341_SLPOUT 0x11 ///< Sleep Out
#define ILI9341_PTLON 0x12  ///< Partial Mode ON
#define ILI9341_NORON 0x13  ///< Normal Display Mode ON

#define ILI9341_RDMODE 0x0A     ///< Read Display Power Mode
#define ILI9341_RDMADCTL 0x0B   ///< Read Display MADCTL
#define ILI9341_RDPIXFMT 0x0C   ///< Read Display Pixel Format
#define ILI9341_RDIMGFMT 0x0D   ///< Read Display Image Format
#define ILI9341_RDSELFDIAG 0x0F ///< Read Display Self-Diagnostic Result

#define ILI9341_INVOFF 0x20   ///< Display Inversion OFF
#define ILI9341_INVON 0x21    ///< Display Inversion ON
#define ILI9341_GAMMASET 0x26 ///< Gamma Set
#define ILI9341_DISPOFF 0x28  ///< Display OFF
#define ILI9341_DISPON 0x29   ///< Display ON

#define ILI9341_CASET 0x2A ///< Column Address Set
#define ILI9341_PASET 0x2B ///< Page Address Set
#define ILI9341_RAMWR 0x2C ///< Memory Write
#define ILI9341_RAMRD 0x2E ///< Memory Read

#define ILI9341_PTLAR 0x30    ///< Partial Area
#define ILI9341_VSCRDEF 0x33  ///< Vertical Scrolling Definition
#define ILI9341_MADCTL 0x36   ///< Memory Access Control
#define ILI9341_VSCRSADD 0x37 ///< Vertical Scrolling Start Address
#define ILI9341_PIXFMT 0x3A   ///< COLMOD: Pixel Format Set

#define ILI9341_FRMCTR1                                                        \
  0xB1 ///< Frame Rate Control (In Normal Mode/Full Colors)
#define ILI9341_FRMCTR2 0xB2 ///< Frame Rate Control (In Idle Mode/8 colors)
#define ILI9341_FRMCTR3                                                        \
  0xB3 ///< Frame Rate control (In Partial Mode/Full Colors)
#define ILI9341_INVCTR 0xB4  ///< Display Inversion Control
#define ILI9341_DFUNCTR 0xB6 ///< Display Function Control

#define ILI9341_PWCTR1 0xC0 ///< Power Control 1
#define ILI9341_PWCTR2 0xC1 ///< Power Control 2
#define ILI9341_PWCTR3 0xC2 ///< Power Control 3
#define ILI9341_PWCTR4 0xC3 ///< Power Control 4
#define ILI9341_PWCTR5 0xC4 ///< Power Control 5
#define ILI9341_VMCTR1 0xC5 ///< VCOM Control 1
#define ILI9341_VMCTR2 0xC7 ///< VCOM Control 2

#define ILI9341_RDID1 0xDA ///< Read ID 1
#define ILI9341_RDID2 0xDB ///< Read ID 2
#define ILI9341_RDID3 0xDC ///< Read ID 3
#define ILI9341_RDID4 0xDD ///< Read ID 4

#define ILI9341_GMCTRP1 0xE0 ///< Positive Gamma Correction
#define ILI9341_GMCTRN1 0xE1 ///< Negative Gamma Correction
//#define ILI9341_PWCTR6     0xFC

extern const uint8_t font6x8[];

void ili9341_init();
void ili9341_set_command(uint8_t cmd);
void ili9341_command_param(uint8_t data);
void ili9341_write_data(void *buffer, int bytes);
void ili9341_start_writing();
void ili9341_stop_writing();
void ili9341_write_data_continuous(void *biffer, int bytes);
#endif
```

ili9341.c

```
#include <stdint.h>
#include <stdio.h>
#include <string.h>
#include "pico/stdlib.h"
#include "ili9341/ili9341.h"

/*

 (pin 1) VCC        5V/3.3V power input
 (pin 2) GND        Ground
 (pin 3) CS         LCD chip select signal, low level enable
 (pin 4) RESET      LCD reset signal, low level reset
 (pin 5) DC/RS      LCD register / data selection signal; high level: register, low level: data
 (pin 6) SDI(MOSI)  SPI bus write data signal
 (pin 7) SCK        SPI bus clock signal
 (pin 8) LED        Backlight control; if not controlled, connect 3.3V always bright
 (pin 9) SDO(MISO)  SPI bus read data signal; optional

 */

ili9341_config_t ili9341_config = {
        .port = spi0,
        .pin_miso = 4,
        .pin_cs = 5,
        .pin_sck = 6,
        .pin_mosi = 7,
        .pin_reset = 8,
        .pin_dc = 9
};

static inline void cs_select() {
    asm volatile("nop \n nop \n nop");
    gpio_put(ili9341_config.pin_cs, 0);  // Active low
    asm volatile("nop \n nop \n nop");
}

static inline void cs_deselect() {
    asm volatile("nop \n nop \n nop");
    gpio_put(ili9341_config.pin_cs, 1);
    asm volatile("nop \n nop \n nop");
}

void ili9341_set_command(uint8_t cmd) {
    cs_select();
    gpio_put(ili9341_config.pin_dc, 0);
    spi_write_blocking(ili9341_config.port, &cmd, 1);
    gpio_put(ili9341_config.pin_dc, 1);
    cs_deselect();
}

void ili9341_command_param(uint8_t data) {
    cs_select();
    spi_write_blocking(ili9341_config.port, &data, 1);
    cs_deselect();
}

inline void ili9341_start_writing() {
    cs_select();
}

void ili9341_write_data(void *buffer, int bytes) {
    cs_select();
    spi_write_blocking(ili9341_config.port, buffer, bytes);
    cs_deselect();
}

void ili9341_write_data_continuous(void *buffer, int bytes) {
    spi_write_blocking(ili9341_config.port, buffer, bytes);
}

inline void ili9341_stop_writing() {
    cs_deselect();
}

void ili9341_init() {
    // This example will use SPI0 at 0.5MHz.
    spi_init(ili9341_config.port, 500 * 1000);
    int baudrate = spi_set_baudrate(ili9341_config.port, 75000 * 1000);

    gpio_set_function(ili9341_config.pin_miso, GPIO_FUNC_SPI);
    gpio_set_function(ili9341_config.pin_sck, GPIO_FUNC_SPI);
    gpio_set_function(ili9341_config.pin_mosi, GPIO_FUNC_SPI);

    // Chip select is active-low, so we'll initialise it to a driven-high state
    gpio_init(ili9341_config.pin_cs);
    gpio_set_dir(ili9341_config.pin_cs, GPIO_OUT);
    gpio_put(ili9341_config.pin_cs, 0);

    // Reset is active-low
    gpio_init(ili9341_config.pin_reset);
    gpio_set_dir(ili9341_config.pin_reset, GPIO_OUT);
    gpio_put(ili9341_config.pin_reset, 1);
    // high = command, low = data
    gpio_init(ili9341_config.pin_dc);
    gpio_set_dir(ili9341_config.pin_dc, GPIO_OUT);
    gpio_put(ili9341_config.pin_dc, 0);


    sleep_ms(10);
    gpio_put(ili9341_config.pin_reset, 0);
    sleep_ms(10);
    gpio_put(ili9341_config.pin_reset, 1);

    ili9341_set_command(0x01);//soft reset
    sleep_ms(100);

    ili9341_set_command(ILI9341_GAMMASET);
    ili9341_command_param(0x01);

    // positive gamma correction
    ili9341_set_command(ILI9341_GMCTRP1);
    ili9341_write_data((uint8_t[15]){ 0x0f, 0x31, 0x2b, 0x0c, 0x0e, 0x08, 0x4e, 0xf1, 0x37, 0x07, 0x10, 0x03, 0x0e,
0x09, 0x00 }, 15);

    // negative gamma correction
    ili9341_set_command(ILI9341_GMCTRN1);
    ili9341_write_data((uint8_t[15]){ 0x00, 0x0e, 0x14, 0x03, 0x11, 0x07, 0x31, 0xc1, 0x48, 0x08, 0x0f, 0x0c, 0x31,
0x36, 0x0f }, 15);

    // memory access control
    ili9341_set_command(ILI9341_MADCTL);
    ili9341_command_param(0x48);

    // pixel format
    ili9341_set_command(ILI9341_PIXFMT);
    ili9341_command_param(0x55);  // 16-bit

    // frame rate; default, 70 Hz
    ili9341_set_command(ILI9341_FRMCTR1);
    ili9341_command_param(0x00);
    ili9341_command_param(0x1B);

    // exit sleep
    ili9341_set_command(ILI9341_SLPOUT);

    // display on
    ili9341_set_command(ILI9341_DISPON);

    //


    // column address set
    ili9341_set_command(ILI9341_CASET);
    ili9341_command_param(0x00);
    ili9341_command_param(0x00);  // start column
    ili9341_command_param(0x00);
    ili9341_command_param(0xef);  // end column -> 239

    // page address set
    ili9341_set_command(ILI9341_PASET);
    ili9341_command_param(0x00);
    ili9341_command_param(0x00);  // start page
    ili9341_command_param(0x01);
    ili9341_command_param(0x3f);  // end page -> 319

    ili9341_set_command(ILI9341_RAMWR);


}

uint16_t swap_bytes(uint16_t color) {
    return (color>>8) | (color<<8);
}
```