classmate

Date 19.6.2019

Page 1

SPI - MASTER

display a simple graphics with varying pixel colors on the 320 x 240 LCD on an ESP-WROVER-KIT.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_system.h"
#include "driver/spi_master.h"
#include "driver/gpio.h"
#include "pretty_effect.h"      ← local

// spi_device_transmit
// spi_device_queue_trans
// spi_device_trans_result
// ILI9341 / ST7789V :

# define PIN_NUM_MISO  25
# define PIN_NUM_MOSI  23
#define PIN_NUM_CLK    19
#define PIN_NUM_CS     22

#define PIN_NUM_DC   21
#define PIN_NUM_RST  18
#define PIN_NUM_BCKL  5
#define PARALLEL_LINES  16
```

```
typedef struct {
 uint8_t cmd;
 uint8_t data[16];
 uint8_t databytes;
} lcd_init_cmd_t;

typedef enum {
  LCD_TYPE_ILI = 1,
  LCD_TYPE_ST,
  LCD_TYPE_MAX,
} type_lcd_t;


DRAM_ATTR static const lcd_init_cmd_t st_init_cmds[] = {
  ...
};

DRAM_ATTR static const lcd_init_cmd_t ili_init_cmds[] = {
  ...
};


void lcd_cmd(spi_device_handle_t spi, const uint8_t cmd) {
  esp_err_t ret;
  spi_transaction_t t;
  memset(&t, 0, sizeof(t));
  t.length = 8;
  t.tx_buffer = &cmd;
  t.user = (void*) 0;
```

```
    ret = spi_device_polling_transmit(spi, &t);
    assert(ret == ESP_OK);
}


void lcd_data(spi_device_handle_t spi, const uint8_t *data, int len) {
    esp_err_t ret;
    spi_transaction_t t;
    if(len==0) return;
    memset(&t, 0, sizeof(t));
    t.length = len*8;
    t.tx_buffer = data;
    t.user = (void*)1;
    ret = spi_device_polling_tranmit(spi, &t);
    asert(ret == ESP_OK);
}


void lcd_spi_pre_transfer_callback(spi_transaction_t *t){
    int dc = (int)t->user;
    gpio_set_level(PIN_NUM_DC, dc);
}


uint32_t lcd_get_id(spi_device_handle_t spi) {
    lcd_cmd(spi, 0x04);
    spi_trancaction_t t;
    memset(&t, 0, sizeof(t));
    t.length = 8*3;
    t.flags = SPI_TRANS_USE_RXDATA;
    t.user  = (void *) 1;
    esp_err_t ret = spi_device_polling_transmit (spi, &t);
    assert (ret ==ESP_OK);
    return *(uint32_t)t.rx_data;
}


void lcd_init (spi_device_handle_t  spi) {
    int cmd=0;
    const lcd_init_cmd_t *lcd_init_cmds;
    gpio_set_direction (PIN_NUM_DC, GPIO_MODE_OUTPUT);
    gpio_set_direction (PIN_NUM_RST, GPIO_MODE_OUTPUT);
    gpio_set_direction (PIN_NUM_BCKL, GPIO_MODE_OUTPUT);
    gpio_set_level (PIN_NUM_RST, 0);
    vTaskDelay (100 / portTICK_RATE_MS);
    gpio_set_level (PIN_NUM_RST, 1);
    vTaskDelay (100 / portTICK_RATE_MS);

    uint32_t lcd_id = lcd_get_id (spi);
    int lcd_detected_type = 0;
    int lcd_type;
    printf ("LCD ID: %08X\n", lcd_id);
    if (lcd_id == 0) {
        lcd_detected_type = LCD_TYPE_ILI;
        printf ("ILI9341 detected.\n");
    } else {
        lcd_detected_type = LCD_TYPE_ST;
        printf ("ST7789V detected.\n");
    }
```

```
#ifdef CONFIG_LCD_TYPE_AUTO
   lcd_type = lcd_detected_type;
#elif defined(CONFIG_LCD_TYPE_ST7789V)
   printf("kconfig: force CONFIG_LCD_TYPE_ST7789V.\n");
   lcd_type = LCD_TYPE_ST;
#elif defined (CONFIG_LCD_TYPE_ILI9341)
   printf("Kconfig: force CONFIG_LCD_TYPE_ILI9341.\n");
   lcd_type = LCD_TYPE_ILI;
#endif
   if(lcd_type == LCD_TYPE_ST) {
      printf("LCD ST7789V initialization.\n");
      lcd_init_cmds = st_init_cmds;
   } else {
      printf("LCD ILI9341 initialization.\n");
      lcd_init_cmds = ili_init_cmds;
   }

   while(lcd_init_cmds[cmd].databytes != 0xFF) {
      lcd_cmd(spi, lcd_init_cmds[cmd].cmd);
                                           lcd_init_cmds[cmd].databytes&0x1F)
      lcd_data(spi, lcd_init_cmds[cmd].data,
      if(lcd_init_cmds[cmd].databytes & 0x80) {
         vTaskDelay(100 / portTICK_RATE_MS);
      cmd++;
   }

   gpio_set_level(PIN_NUM_BCKL, 0);
}
```

```
static void send_lines (spi_device_handle_t spi, int ypos, uint16_t * linedata) {
    esp_err_t ret;
    int x;
    static spi_transaction_t trans[6];
    for (x=0; x<6; x++) {
      memset (&trans[x], 0, sizeof(spi_transaction_t));
      if ((x&1) == 0) {
         trans[x]. length = 8;
         trans [x]. user = (void *) 0;
      } else {
         trans [x]. length = 8 * 4;
         trans [x]. user = (void *) 1;
      }

      trans[x] flags = SPI_TRANS_USE_TX DATA;
    }

     trans [0 1 2 3] ...
     trans [4]. tx_data [0] = 0x2C;
     trans [5]. tx_buffer = linedata;
     trans[5]. length = 320 x 2 x 8 x PARALLEL_LINES;
     trans [5]. flags = 0;
     for (x = 0; x < 6; x++) {
       ret = spi_device_queue_trans (spi, &trans[x], portMAX_DELAY);
       assert (ret == ESP_OK);
     }
   }
```

in memory
x stack

```
Static void send_line_finish (spi_device_handle_t spi) {
  spi_transaction_t *trans;
  esp_err_t ret;
  for (int x=0; x<6; x++) {
    ret = spi_device_get_trans_result(spi, &rtrans, portMAX_DELAY
    assert(ret== ESP_OK);
  }
}



Static void display_pretty_colors (spi_device_handle_t spi) {
  uint16_t *lines[2];
  for( int i=0; i<2, i++) {
                                                          MALLOC_CAP_DMA);
    lines [i] = heap_caps_malloc (320*PARALLEL_LINES * sizeof(uint16_t)
    assert(lines[i] !=NULL);
  }
  int frame = 0;
  int sending_line = -1;
  int calc_line = 0;

  while (1) {
   frame ++;
   for(int y=0; y<240; y+= PARALLEL_LINES) {
     pretty_effect_calc_lines( lines[calc_line], y, frame, PARALLEL_LINES)
                              -1)
     if (sending_line != ~~calc_line~~ send_line_finish(spi);
     Sending_line = calc_line;
     calc_line = (calc_line == 1) ? 0: 1;
     Send_lines (spi, y, lines[sending_line]);
   }
 }
}
```

classmate
Date 19.6.2019
Page 8

STORAGE WIFI OTA

OBJECTIVE : MOSQUITTO SERVER · MQTT HELLO WORLD
ECLIPSE PUBLISH — SUBSCRIBE

```
void app_main () {
  esp_err_t ret;
  spi_device_handle_t spi;
  spi_bus_config_t buscfg = {
     .miso_io_num = PIN_NUM_MISO,
     .mosi_io_num = PIN_NUM_MOSI,
     .sclk_io_num = PIN_NUM_CLK,
     .quadwp_io_num = -1,
     .quadhd_io_num = -1,
     .max_transfer_sz = PARALLEL_LINES*320*2+8,
  };
  spi_device_interface_config_t devcfg = {
#ifdef CONFIG_LCD_OVERCLOCK
    .clock_speed_hz = 26*1000*1000,
#else
    .clock_speed_hz = 10*1000*1000,
#endif
    .mode = 0,
    .spics_io_num = PIN_NUM_CS,
    .queue_size = 7,
    .pre_cb = lcd_spi_pre_transfer_callback,
  };
  ret = spi_bus_initialize(HSPI_HOST, &buscfg, 1);
  ESP_ERROR_CHECK(ret);
ret = spi_bus_add_device(HSPI_HOST, &devcfg, &spi);
ESP_ERROR_CHECK(ret);
lcd_init(spi);
ret = pretty_effect_init();
ESP_ERROR_CHECK(ret);
display_pretty_colors(spi);
}
```