```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

#include "freertos/FreeRTOS.h"
#include "freertos/event_groups.h"

#include "esp_wifi.h"
#include "esp_system.h"
#include "esp_event.h"
#include "esp_event_loop.h"
#include "esp_log.h"
#include "driver/spi_master.h"
#include "soc/gpio_struct.h"
#include "driver/gpio.h"

#include "mfrc.h"
//#include "network.h"  // for myEventGroup ...
#include "spi.h"
#include "heap_alloc_caps.h"

static const char *TAG = "SPI";

typedef struct {
     TaskHandle_t       xHandle;
     spi_device_handle_t spi;
     spi_config_t       *config;
     int        dc;      // current value of D/C
} pvSPI_t;

static int
spi_transfer(pvSPI_t *pv, char *data, int len, int dc)
{
     esp_err_t ret;
     spi_transaction_t trans, *t = &trans;
     char recvbuff[16];

     ESP_LOGI(TAG, "spi_transfer: entered");
     ESP_LOGI(TAG, "\tdata=%p 0x%02x len=%d, dc=%d",  data,
data[0],len, dc);
     ESP_LOGI(TAG, "pv=%p", pv);
     ESP_LOGI(TAG, "\tdevice='%s'", pv->config->devname);

     memset(t, 0, sizeof(spi_transaction_t));   // Zero out the
transaction
     //t->command = data[0];
     //t.flags = SPI_TRANS_MODE_DIO;
     t->length = len * 8;       // len is in bytes, length is in
bits
     if(len <= 4) {
          ESP_LOGI(TAG, "doing short xmit");
          t->flags = SPI_TRANS_USE_TXDATA;
          t->tx_data[0] = data[0];
     }
```

```
	else
		t->tx_buffer = data;
	pv->dc = dc;
	ESP_LOGI(TAG, "calling  spi_device_transmit ...");
	ret = spi_device_transmit(pv->spi, t);  //Transmit!
	ESP_LOGI(TAG, "\tret=%d", ret);
	return ret == ESP_OK;
}

/*
 | This function is called (in irq context!) just before a
transmission starts. It will
 | set the D/C line to the value indicated in the user field.
 */
static void
_spi_pre_transfer_callback(spi_transaction_t *t)
{
	pvSPI_t *pv= (pvSPI_t *) t->user;
	ESP_LOGI(TAG, "_spi_pre_transfer_callback: %d->%d", pv->config-
>pins[gpio_DC], pv->dc);

	gpio_set_level(pv->config->pins[gpio_DC], pv->dc);
}

static int
dev_spi_init(void *pvParams)
{
	pvSPI_t *pv = pvParams;
	ESP_LOGI(TAG, "dev_spi_init: rst pin is: %d", pv->config-
>pins[gpio_RST]);

	/*
	 | toggle the reset
	 */
	if(pv->config->pins[gpio_RST] != -1) {
		gpio_set_level(pv->config->pins[gpio_RST], 0);
		msDelay(100);
		gpio_set_level(pv->config->pins[gpio_RST], 1);
		msDelay(100);
	}

	return 0;
}


static void
spiTask(void *pvParams)
{
	pvSPI_t *pv = pvParams;

	ESP_LOGI(TAG, "spiTask: entered for device: %s", pv->config-
>devname);

	while(dev_spi_init(pv) != 0) {
```

```
            ESP_LOGE(TAG, "failed to init device '%s'", pv->config-
>devname);
            msDelay(5000);
        }

        ESP_LOGI(TAG, "spiTask: '%s': inited ok", pv->config->devname);

        for(;;) {
            msDelay(10000);
        }
}

void *
spiInit(spi_config_t *config)
{
        pvSPI_t    *pv;

        esp_log_level_set(TAG, ESP_LOG_DEBUG);
        ESP_LOGI(TAG, "spiInit: entered");

        pv = calloc(1, sizeof(pvSPI_t));
        if(pv == NULL) {
            ESP_LOGE(TAG, "no memory available!");
            goto bad;
        }

        pv->config = config;

        esp_err_t ret;
        spi_bus_config_t buscfg = {
            .quadwp_io_num        = -1,
            .quadhd_io_num        = -1
        };
        buscfg.miso_io_num = config->pins[gpio_MISO];
        buscfg.mosi_io_num = config->pins[gpio_MOSI];
        buscfg.sclk_io_num = config->pins[gpio_CLK];

        spi_device_interface_config_t devcfg = {
            //.command_bits = 8,
            //.address_bits = 64,
            .clock_speed_hz       = 10000000,     // Clock out at 10
MHz
            .mode                 = 0,            // SPI mode 0
            .queue_size           = 7,            // We want to be
able to queue 7 transactions at a time
            .pre_cb               = _spi_pre_transfer_callback,
        };
        devcfg.spics_io_num = config->pins[gpio_CS];
        /*
         | Initialize the SPI bus
         */
        ret = spi_bus_initialize(HSPI_HOST, &buscfg, 1);
        if(ret != ESP_OK) {
            ESP_LOGE(TAG, "spi_bus_initialize failed: %d", ret);
```

```
          goto bad;
     }
     /*
      | Attach the device to the SPI bus
      */
     ret = spi_bus_add_device(HSPI_HOST, &devcfg, &pv->spi);
     if(ret != ESP_OK) {
          ESP_LOGE(TAG, "spi_bus_add device failed: %d", ret);
          goto bad;
     }

     BaseType_t err = xTaskCreate(&spiTask, "spiTask",
                                  configMINIMAL_STACK_SIZE,
                                  (void *)pv,
                                  5,    // priority
                                  &pv->xHandle);

     if(err != pdPASS) {
          ESP_LOGE(TAG, "failed to create task!");
     bad:
          if(pv)
               free(pv);
          return NULL;
     }
     /*
      | Initialize non-SPI GPIOs
      */
     if(config->pins[gpio_DC] != -1)
          gpio_set_direction(config->pins[gpio_DC],
GPIO_MODE_OUTPUT);
     if(config->pins[gpio_RST] != -1)
          gpio_set_direction(config->pins[gpio_RST],
GPIO_MODE_OUTPUT);

     ESP_LOGI(TAG, "spiInit OK");

     return (void *)pv;
}

int
spiCmd(void *pv, char cmd)
{
     int res;

     // make sure DC is 0/low
     res = spi_transfer(pv, &cmd, 1, 0);
     return res;
}

int
spiDat(void *pv, char *data, int len)
{
     // make sure DC is 1/high
     return spi_transfer(pv, data, len, 1);
```

```
}

#include "ssd1306.h"
int
spiTest()
{
     return ssd1306Test();
}
```