#include <stdio.h>

#include <time.h>

#include <sys/time.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "freertos/event\_groups.h"

#include "driver/gpio.h"

#include "sdkconfig.h"

#include "esp\_log.h"

#include "freertos/queue.h"

#include "ssd1306.h"

#include "timeMgmt.h"

#include <driver/adc.h>

#include "mqtt.h"

#include "nvs\_flash.h"

#include "wifi.h"

#define I2C\_MASTER\_SCL\_IO CONFIG\_I2C\_MASTER\_SCL               /\*!< gpio number for I2C master clock \*/

#define I2C\_MASTER\_SDA\_IO CONFIG\_I2C\_MASTER\_SDA

volatile uint8\_t count = 0;

// #define I2C\_APB\_CLK\_FREQ  APB\_CLK\_FREQ

// #define I2C\_SCLK\_SRC\_FLAG\_FOR\_NOMAL       (0)         /\*!< Any one clock source that is available for the specified frequency may be choosen\*/

int debounceTime = 4;

int OUTER\_GPIO\_PIN = GPIO\_NUM\_18;

int INNER\_GPIO\_PIN = GPIO\_NUM\_19;

// QueueHandle\_t xQueueLeftIn, xQueueRightIn;

enum State{

    IDLE,

    FIRSTONLY,

    BOTH,

    SECONDONLY

};

bool innerFirst = true;

enum State state = IDLE;

void initDisplay(){

    ssd1306\_128x64\_i2c\_init();

    ssd1306\_setFixedFont(ssd1306xled\_font6x8);

}

static const char \*pcTextForTask1 = "Counter Task is running\r\n";

static const char \*pcTextForTask2 = "Publish Task is running\r\n";

long long lastInnerMilliseconds = 0;

long long lastOuterMilliseconds = 0;

void handleInnerBarrier()

{

    struct timeval te;

    gettimeofday(&te, NULL); // get current time

    long long milliseconds = te.tv\_sec\*1000LL + te.tv\_usec/1000; // calculate milliseconds

    if(milliseconds > lastInnerMilliseconds + debounceTime){

        lastInnerMilliseconds = milliseconds;

        if(gpio\_get\_level(INNER\_GPIO\_PIN) == 0){

            if(state == FIRSTONLY && innerFirst){

                state = IDLE;

                ets\_printf("going to state: %d \tgoing back and idle\tunbreaching inner\n", state);

                ets\_printf("----------------------------------------------\n");

            }

            else if(state == BOTH && !innerFirst){

                state = FIRSTONLY;

                count++;

                ets\_printf("going to state BOTH: %d \tgoing back and increasing count\tunbreaching inner\n", state);

            }

            else if(state == BOTH && innerFirst){

                state = SECONDONLY;

                if(count>0) count--;

                ets\_printf("going to state: %d \tgoing farther and decreasing count\tunbreaching inner\n", state);

            }

            else if(state == SECONDONLY){

                state = IDLE;

                //count++;

                ets\_printf("going to state: %d \tunbreaching inner\n", state);

                ets\_printf("----------------------------------------------\n");

            }

        }

        if(gpio\_get\_level(INNER\_GPIO\_PIN) == 1){

            if(state == IDLE){

                state = FIRSTONLY;

                innerFirst = true;

                ets\_printf("going to state: %d \tInner breached first\tbreaching inner\n", state);

            }

            else if(state == FIRSTONLY){

                state = BOTH;

                ets\_printf("going to state: %d \tgoing farther\tbreaching inner\n", state);

            }

            else if(state == SECONDONLY && innerFirst){

                state = BOTH;

                ets\_printf("going to state: %d \tgoing back\tbreaching inner\n", state);

            }

        }

    }

}

void handleOuterBarrier()

{

    struct timeval te;

    gettimeofday(&te, NULL); // get current time

    long long milliseconds = te.tv\_sec\*1000LL + te.tv\_usec/1000; // calculate milliseconds

    if(milliseconds > lastOuterMilliseconds + debounceTime){

        lastOuterMilliseconds = milliseconds;

        if(gpio\_get\_level(OUTER\_GPIO\_PIN) == 0){

            INNER\_PIN = gpio\_get\_level(INNER\_GPIO\_PIN);

            if (INNER\_PIN == 1 && innerFirst) {

                state = FIRSTONLY;

            }

            if (INNER\_PIN == 1 && !innerFirst) {

                state = SECONDONLY;

            }

            if (INNER\_PIN == 0) {

                state = IDLE;

            }

            if(state == FIRSTONLY && !innerFirst){

                state = IDLE;

                ets\_printf("going to state: %d \tgoing back and idle\tunbreaching outer\n", state);

                ets\_printf("----------------------------------------------\n");

            }

            else if(state == BOTH && innerFirst){

                state = FIRSTONLY;

                if (count>0) count--;

                ets\_printf("going to state BOTH: %d \tgoing back and decreasing count\tunbreaching outer\n", state);

            }

            else if(state == BOTH && !innerFirst){

                state = SECONDONLY;

                ets\_printf("going to state BOTH: %d \tgoing farther and increasing count\tunbreaching outer\n", state);

                count++;

            }

            else if(state == SECONDONLY){

                state = IDLE;

                // if(count>0) count--;

                ets\_printf("going to state: %d \tdecreasing count\tunbreaching outer\n", state);

                ets\_printf("----------------------------------------------\n");

            }

        }

        if(gpio\_get\_level(OUTER\_GPIO\_PIN) == 1){

            if(state == IDLE){

                state = FIRSTONLY;

                innerFirst = false;

                ets\_printf("going to state: %d \tOuter breached first\tbreaching outer\n", state);

            }

            else if(state == FIRSTONLY){

                state = BOTH;

                ets\_printf("going to state: %d \tgoing farther\tbreaching outer\n", state);

            }

            else if(state == SECONDONLY && innerFirst){

                state = BOTH;

                ets\_printf("going to state: %d \tgoing back\tbreaching outer\n", state);

            }

        }

    }

}

void text(){

    ssd1306\_clearScreen();

    //ssd1306\_printFixedN(0, 0, "Normal text", STYLE\_NORMAL, 1);

    char countStr[21]; //for x64 machines

    sprintf(countStr, "%d", count);

    //ESP\_LOGI("OLED", "TRYING TO PRINT");

    ssd1306\_printFixedN(0, 0, "G9", STYLE\_NORMAL, 1);

    time\_t current\_time;

    struct tm \*time\_info;

    char strftime\_buf[9];  // space for "HH:MM:SS\0"

    time(&current\_time);

    time\_info = localtime(&current\_time);

    strftime(strftime\_buf, sizeof(strftime\_buf), "%H:%M", time\_info);

    ssd1306\_printFixedN(60, 0, strftime\_buf, STYLE\_NORMAL, 1);

    ssd1306\_printFixedN(0, 50, countStr, STYLE\_NORMAL, 1);

    ssd1306\_printFixedN(100, 50, "0", STYLE\_NORMAL, 1);

}

void vCounterTask (void \*pvParameters){

    char \*pcTaskName;

    pcTaskName = ( char \* ) pvParameters;

    ets\_printf(pcTaskName);

    ESP\_ERROR\_CHECK(gpio\_set\_intr\_type(INNER\_GPIO\_PIN, GPIO\_INTR\_ANYEDGE));

    ESP\_ERROR\_CHECK(gpio\_isr\_handler\_add(INNER\_GPIO\_PIN, handleInnerBarrier, (void\*) INNER\_GPIO\_PIN));

    ESP\_ERROR\_CHECK(gpio\_set\_intr\_type(OUTER\_GPIO\_PIN, GPIO\_INTR\_ANYEDGE));

    ESP\_ERROR\_CHECK(gpio\_isr\_handler\_add(OUTER\_GPIO\_PIN, handleOuterBarrier,(void\*) OUTER\_GPIO\_PIN));

    for(;;){

        vTaskDelay(10/portTICK\_PERIOD\_MS);

    }

}

void vPublishTask(void \*parameters){

    for(;;){

        time\_t now = 0;

        struct tm \*time\_info;

        time(&now);

        //Check if shortly after 00:00h, if yes: reset count

        char strftime\_buf[6];  // space for "HH:MM\0"

        time\_info = localtime(&now);

        strftime(strftime\_buf, sizeof(strftime\_buf), "%H:%M", time\_info);

        if((now %(24\*3600)) >= 0 && (now %(24\*3600)) <= 400){

            count = 0;

        }

        //publish current count

        char msg[256];

        sprintf(msg, "{\"username\":\"group9\",\"device\_id\":\"7\",\"count\":%d,\"timestamp\":%lu000}", count, now);

        esp\_mqtt\_client\_publish(mqttClient, "11\_7", msg, strlen(msg), 1,0);

        vTaskDelay(300000/portTICK\_PERIOD\_MS);

    }

}

void showRoomState(){

    ets\_printf("oled task running");

    for(;;){

        text();

        vTaskDelay(1000 / portTICK\_PERIOD\_MS);

    }

}

void IRAM\_ATTR outISR(void\* arg){

    ets\_printf("Interrupt OUT.\n");

}

void app\_main(void){

    ets\_printf("running app....\n");

    //esp\_log\_level\_set("BLINK", ESP\_LOG\_ERROR);

    esp\_log\_level\_set("BLINK", ESP\_LOG\_INFO);

    //ESP\_ERROR\_CHECK(gpio\_set\_direction(INNER\_GPIO\_PIN, GPIO\_MODE\_INPUT));

    //ESP\_ERROR\_CHECK(gpio\_set\_direction(OUTER\_GPIO\_PIN, GPIO\_MODE\_INPUT));

//  ESP\_ERROR\_CHECK(gpio\_pullup\_dis(INNER\_GPIO\_PIN));

//  ESP\_ERROR\_CHECK(gpio\_pulldown\_dis(INNER\_GPIO\_PIN));

    //ESP\_ERROR\_CHECK(gpio\_pulldown\_en(INNER\_GPIO\_PIN));

    //ESP\_ERROR\_CHECK(gpio\_pullup\_dis(INNER\_GPIO\_PIN));

    //ESP\_ERROR\_CHECK(gpio\_pulldown\_en(OUTER\_GPIO\_PIN));

    //ESP\_ERROR\_CHECK(gpio\_pullup\_dis(OUTER\_GPIO\_PIN));

    ESP\_ERROR\_CHECK(gpio\_set\_direction(INNER\_GPIO\_PIN, GPIO\_MODE\_INPUT));

    ESP\_ERROR\_CHECK(gpio\_set\_direction(OUTER\_GPIO\_PIN, GPIO\_MODE\_INPUT));

    ESP\_ERROR\_CHECK(gpio\_set\_level(OUTER\_GPIO\_PIN, 0));

    ESP\_ERROR\_CHECK(gpio\_set\_level(INNER\_GPIO\_PIN, 0));

//  ESP\_ERROR\_CHECK(gpio\_pulldown\_en(INNER\_GPIO\_PIN));

//  ESP\_ERROR\_CHECK(gpio\_pulldown\_en(OUTER\_GPIO\_PIN));

    ESP\_ERROR\_CHECK(gpio\_pullup\_dis(OUTER\_GPIO\_PIN));

    ESP\_ERROR\_CHECK(gpio\_pulldown\_dis(OUTER\_GPIO\_PIN));

    ESP\_ERROR\_CHECK(gpio\_pullup\_dis(INNER\_GPIO\_PIN));

    ESP\_ERROR\_CHECK(gpio\_pulldown\_dis(INNER\_GPIO\_PIN));

    ESP\_ERROR\_CHECK(gpio\_install\_isr\_service(ESP\_INTR\_FLAG\_IRAM));

    /\* Create a queue capable of containing 10 unsigned long values. \*/

    // xQueueRightIn = xQueueCreate( 10, sizeof( unsigned long ) );

    // xQueueLeftIn = xQueueCreate( 10, sizeof( unsigned long ) );

    //void ssd1306\_i2cInit\_Embedded(int8\_t scl, int8\_t sda, uint8\_t sa);

    esp\_err\_t ret = nvs\_flash\_init();

    if (ret == ESP\_ERR\_NVS\_NO\_FREE\_PAGES || ret == ESP\_ERR\_NVS\_NEW\_VERSION\_FOUND) {

        ESP\_ERROR\_CHECK(nvs\_flash\_erase());

        ret = nvs\_flash\_init();

    }

    ESP\_ERROR\_CHECK(ret);

    initWifi();

    initSNTP();

    initMQTT();

    initDisplay();

    text();

    xTaskCreate( showRoomState, "Task 3", 10000, NULL, 1, NULL);

    xTaskCreate( vPublishTask, "Task 2", 100000, (void\*)pcTextForTask2, 100, NULL);

    xTaskCreate( vCounterTask, "Task 1", 1000, (void\*)pcTextForTask1, 100, NULL);

}