COMPLETED CODE [LAYOUT II]

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/unistd.h>
#include <sys/stat.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/queue.h"
#include "freertos/event_groups.h"
#include "esp_system.h"
#include "esp_wifi.h"
#include "esp_event.h"
#include "esp_log.h"
#include "nvs_flash.h"
#include "esp_vfs.h"
#include "esp_spiffs.h"
#include "esp_sntp.h"
#include "mdns.h"
#include "lwip/dns.h"
#include "driver/gpio.h"
#include "esp_camera.h"
#include "camera_pin.h"
#include "cmd.h"

#if (ESP_IDF_VERSION >= ESP_IDF_VERSION_VAL(5, 0, 0))
#define sntp_setoperatingmode esp_sntp_setoperatingmode
#define sntp_setservername esp_sntp_setservername
#define sntp_init esp_sntp_init
#endif

static EventGroupHandle_t s_wifi_event_group;
#define WIFI_CONNECTED_BIT BIT0
#define WIFI_FAIL_BIT BIT1
static const char *TAG = "MAIN";
static int s_retry_num = 0;

QueueHandle_t xQueueCmd;
QueueHandle_t xQueueHttp;
QueueHandle_t xQueueRequest;
```

```
camera_config_t camera_config = {
    .pin_pwdn = CAM_PIN_PWDN,
    .pin_reset = CAM_PIN_RESET,
    .pin_xclk = CAM_PIN_XCLK,
    .pin_sscb_sda = CAM_PIN_SIOD,
    .pin_sscb_scl = CAM_PIN_SIOC,
    .pin_d7 = CAM_PIN_D7,
    .pin_d6 = CAM_PIN_D6,
    .pin_d5 = CAM_PIN_D5,
    .pin_d4 = CAM_PIN_D4,
    .pin_d3 = CAM_PIN_D3,
    .pin_d2 = CAM_PIN_D2,
    .pin_d1 = CAM_PIN_D1,
    .pin_d0 = CAM_PIN_D0,
    .pin_vsync = CAM_PIN_VSYNC,
    .pin_href = CAM_PIN_HREF,
    .pin_pclk = CAM_PIN_PCLK,
    .xclk_freq_hz = 20000000,
    .ledc_timer = LEDC_TIMER_0,
    .ledc_channel = LEDC_CHANNEL_0,
    .pixel_format = PIXFORMAT_JPEG,
    .frame_size = FRAMESIZE_VGA,
    .jpeg_quality = 12,
    .fb_count = 1
};

static esp_err_t init_camera(int framesize) {
    camera_config.frame_size = framesize;
    esp_err_t err = esp_camera_init(&camera_config);
    if (err != ESP_OK) {
        ESP_LOGE(TAG, "Camera Init Failed");
        return err;
    }
    return ESP_OK;
}

static esp_err_t camera_capture(char * FileName, size_t *pictureSize) {
    for(int i=0;i<1;i++) {
        camera_fb_t * fb = esp_camera_fb_get();
        ESP_LOGI(TAG, "fb->len=%d", fb->len);
        esp_camera_fb_return(fb);
    }
    camera_fb_t * fb = esp_camera_fb_get();
```

```
    if (!fb) {
        ESP_LOGE(TAG, "Camera Capture Failed");
        return ESP_FAIL;
    }
    FILE* f = fopen(FileName, "wb");
    if (f == NULL) {
        ESP_LOGE(TAG, "Failed to open file for writing");
        return ESP_FAIL;
    }
    fwrite(fb->buf, fb->len, 1, f);
    ESP_LOGI(TAG, "fb->len=%d", fb->len);
    *pictureSize = (size_t)fb->len;
    fclose(f);
    esp_camera_fb_return(fb);
    return ESP_OK;
}

static void event_handler(void* arg, esp_event_base_t event_base,
int32_t event_id, void* event_data) {
    if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_START) {
        esp_wifi_connect();
    } else if (event_base == WIFI_EVENT && event_id ==
WIFI_EVENT_STA_DISCONNECTED) {
        if (s_retry_num < CONFIG_ESP_MAXIMUM_RETRY) {
            esp_wifi_connect();
            s_retry_num++;
            ESP_LOGI(TAG, "retry to connect to the AP");
        } else {
            xEventGroupSetBits(s_wifi_event_group, WIFI_FAIL_BIT);
        }
        ESP_LOGI(TAG,"connect to the AP fail");
    } else if (event_base == IP_EVENT && event_id ==
IP_EVENT_STA_GOT_IP) {
        ip_event_got_ip_t* event = (ip_event_got_ip_t*) event_data;
        ESP_LOGI(TAG, "got ip:" IPSTR, IP2STR(&event->ip_info.ip));
        s_retry_num = 0;
        xEventGroupSetBits(s_wifi_event_group, WIFI_CONNECTED_BIT);
    }
}

void wifi_init_sta() {
    s_wifi_event_group = xEventGroupCreate();
    ESP_LOGI(TAG,"ESP-IDF esp_netif");
```

```
    ESP_ERROR_CHECK(esp_netif_init());
    ESP_ERROR_CHECK(esp_event_loop_create_default());
    esp_netif_t *netif = esp_netif_create_default_wifi_sta();
    assert(netif);

#if CONFIG_STATIC_IP
    ESP_LOGI(TAG,
"CONFIG_STATIC_IP_ADDRESS=[%s]",CONFIG_STATIC_IP_ADDRESS);
    ESP_LOGI(TAG,
"CONFIG_STATIC_GW_ADDRESS=[%s]",CONFIG_STATIC_GW_ADDRESS);
    ESP_LOGI(TAG,
"CONFIG_STATIC_NM_ADDRESS=[%s]",CONFIG_STATIC_NM_ADDRESS);
    ESP_ERROR_CHECK(esp_netif_dhcpc_stop(netif));
    ESP_LOGI(TAG, "Stop DHCP Services");
    esp_netif_ip_info_t ip_info;
    memset(&ip_info, 0 , sizeof(esp_netif_ip_info_t));
    ip_info.ip.addr = ipaddr_addr(CONFIG_STATIC_IP_ADDRESS);
    ip_info.netmask.addr = ipaddr_addr(CONFIG_STATIC_NM_ADDRESS);
    ip_info.gw.addr = ipaddr_addr(CONFIG_STATIC_GW_ADDRESS);;
    esp_netif_set_ip_info(netif, &ip_info);
    ip_addr_t d;
    d.type = IPADDR_TYPE_V4;
    d.u_addr.ip4.addr = 0x08080808;
    dns_setserver(0, &d);
    d.u_addr.ip4.addr = 0x08080404;
    dns_setserver(1, &d);
#endif

    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK(esp_wifi_init(&cfg));
    ESP_ERROR_CHECK(esp_event_handler_register(WIFI_EVENT,
ESP_EVENT_ANY_ID, &event_handler, NULL));
    ESP_ERROR_CHECK(esp_event_handler_register(IP_EVENT,
IP_EVENT_STA_GOT_IP, &event_handler, NULL));
    wifi_config_t wifi_config = {
        .sta = {
            .ssid = CONFIG_ESP_WIFI_SSID,
            .password = CONFIG_ESP_WIFI_PASSWORD
        },
    };
    ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA));
    ESP_ERROR_CHECK(esp_wifi_set_config(ESP_IF_WIFI_STA,
&wifi_config));
```

```
    ESP_ERROR_CHECK(esp_wifi_start());
    ESP_LOGI(TAG, "wifi_init_sta finished.");
    EventBits_t bits = xEventGroupWaitBits(s_wifi_event_group,
WIFI_CONNECTED_BIT | WIFI_FAIL_BIT, pdFALSE, pdFALSE, portMAX_DELAY);
    if (bits & WIFI_CONNECTED_BIT) {
        ESP_LOGI(TAG, "connected to ap SSID:%s password:%s",
CONFIG_ESP_WIFI_SSID, CONFIG_ESP_WIFI_PASSWORD);
    } else if (bits & WIFI_FAIL_BIT) {
        ESP_LOGI(TAG, "Failed to connect to SSID:%s, password:%s",
CONFIG_ESP_WIFI_SSID, CONFIG_ESP_WIFI_PASSWORD);
    } else {
        ESP_LOGE(TAG, "UNEXPECTED EVENT");
    }
    vEventGroupDelete(s_wifi_event_group);
}

void initialise_mdns(void) {
    ESP_ERROR_CHECK(mdns_init());
    ESP_ERROR_CHECK(mdns_hostname_set(CONFIG_MDNS_HOSTNAME));
    ESP_LOGI(TAG, "mdns hostname set to: [%s]", CONFIG_MDNS_HOSTNAME);
#if 0
    ESP_ERROR_CHECK(mdns_instance_name_set("ESP32 with mDNS"));
#endif
}

esp_err_t mountSPIFFS(char * partition_label, char * base_path) {
    ESP_LOGI(TAG, "Initializing SPIFFS file system");
    esp_vfs_spiffs_conf_t conf = {
        .base_path = base_path,
        .partition_label = partition_label,
        .max_files = 8,
        .format_if_mount_failed = true
    };
    esp_err_t ret = esp_vfs_spiffs_register(&conf);
    if (ret != ESP_OK) {
        if (ret == ESP_FAIL) {
            ESP_LOGE(TAG, "Failed to mount or format filesystem");
        } else if (ret == ESP_ERR_NOT_FOUND) {
            ESP_LOGE(TAG, "Failed to find SPIFFS partition");
        } else {
            ESP_LOGE(TAG, "Failed to initialize SPIFFS (%s)",
esp_err_to_name(ret));
        }
```

```
        return ret;
    }
    size_t total = 0, used = 0;
    ret = esp_spiffs_info(partition_label, &total, &used);
    if (ret != ESP_OK) {
        ESP_LOGE(TAG, "Failed to get SPIFFS partition information
(%s)", esp_err_to_name(ret));
    } else {
        ESP_LOGI(TAG, "Partition size: total: %d, used: %d", total,
used);
    }
    ESP_LOGI(TAG, "Mount SPIFFS filesystem");
    return ret;
}

#if CONFIG_REMOTE_IS_VARIABLE_NAME
void time_sync_notification_cb(struct timeval *tv) {
    ESP_LOGI(TAG, "Notification of a time synchronization event");
}

static void initialize_sntp(void) {
    ESP_LOGI(TAG, "Initializing SNTP");
    sntp_setoperatingmode(SNTP_OPMODE_POLL);
    ESP_LOGI(TAG, "Your NTP Server is %s", CONFIG_NTP_SERVER);
    sntp_setservername(0, CONFIG_NTP_SERVER);
    sntp_set_time_sync_notification_cb(time_sync_notification_cb);
    sntp_init();
}

static esp_err_t obtain_time(void) {
    initialize_sntp();
    int retry = 0;
    const int retry_count = 10;
    while (sntp_get_sync_status() == SNTP_SYNC_STATUS_RESET && ++retry
< retry_count) {
        ESP_LOGI(TAG, "Waiting for system time to be set... (%d/%d)",
retry, retry_count);
        vTaskDelay(2000 / portTICK_PERIOD_MS);
    }
    if (retry == retry_count) return ESP_FAIL;
    return ESP_OK;
}
#endif
```

```
void http_post_task(void *pvParameters);
#if CONFIG_SHUTTER_ENTER
void keyin(void *pvParameters);
#endif
#if CONFIG_SHUTTER_GPIO
void gpio(void *pvParameters);
#endif
#if CONFIG_SHUTTER_TCP
void tcp_server(void *pvParameters);
#endif
#if CONFIG_SHUTTER_UDP
void udp_server(void *pvParameters);
#endif
void http_task(void *pvParameters);

void app_main(void) {
    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);
    wifi_init_sta();
    initialise_mdns();

#if CONFIG_REMOTE_IS_VARIABLE_NAME
    ESP_LOGI(TAG, "Connecting to WiFi and getting time over NTP.");
    ret = obtain_time();
    if(ret != ESP_OK) {
        ESP_LOGE(TAG, "Fail to getting time over NTP.");
        return;
    }
    time_t now;
    struct tm timeinfo;
    char strftime_buf[64];
    time(&now);
    now = now + (CONFIG_LOCAL_TIMEZONE*60*60);
    localtime_r(&now, &timeinfo);
    strftime(strftime_buf, sizeof(strftime_buf), "%c", &timeinfo);
    ESP_LOGI(TAG, "The current date/time is: %s", strftime_buf);
#endif
```

```
    ESP_LOGI(TAG, "Initializing SPIFFS");
    char *partition_label = "storage";
    char *base_path = "/spiffs";
    ret = mountSPIFFS(partition_label, base_path);
    if (ret != ESP_OK) return;

#if CONFIG_ENABLE_FLASH
    gpio_reset_pin(CONFIG_GPIO_FLASH);
    gpio_set_direction(CONFIG_GPIO_FLASH, GPIO_MODE_OUTPUT);
    gpio_set_level(CONFIG_GPIO_FLASH, 0);
#endif

    xQueueCmd = xQueueCreate(1, sizeof(CMD_t));
    configASSERT(xQueueCmd);
    xQueueRequest = xQueueCreate(1, sizeof(REQUEST_t));
    configASSERT(xQueueRequest);
    xQueueHttp = xQueueCreate(10, sizeof(HTTP_t));
    configASSERT(xQueueHttp);
    xTaskCreate(&http_post_task, "POST", 4096, NULL, 5, NULL);

#if CONFIG_SHUTTER_ENTER
#define SHUTTER "Keybord Enter"
    xTaskCreate(keyin, "KEYIN", 1024*4, NULL, 2, NULL);
#endif
#if CONFIG_SHUTTER_GPIO
#define SHUTTER "GPIO Input"
    xTaskCreate(gpio, "GPIO", 1024*4, NULL, 2, NULL);
#endif
#if CONFIG_SHUTTER_TCP
#define SHUTTER "TCP Input"
    xTaskCreate(tcp_server, "TCP", 1024*4, NULL, 2, NULL);
#endif
#if CONFIG_SHUTTER_UDP
#define SHUTTER "UDP Input"
    xTaskCreate(udp_server, "UDP", 1024*4, NULL, 2, NULL);
#endif

#if CONFIG_SHUTTER_HTTP
#define SHUTTER "HTTP Request"
#endif

    esp_netif_ip_info_t ip_info;
```

```
ESP_ERROR_CHECK(esp_netif_get_ip_info(esp_netif_get_handle_from_ifkey("
WIFI_STA_DEF"), &ip_info));
    char cparam0[64];
    sprintf(cparam0, IPSTR, IP2STR(&ip_info.ip));
    ESP_LOGI(TAG, "cparam0=[%s]", cparam0);
    xTaskCreate(http_task, "HTTP", 1024*6, (void *)cparam0, 2, NULL);

#if CONFIG_FRAMESIZE_VGA
    int framesize = FRAMESIZE_VGA;
    #define FRAMESIZE_STRING "640x480"
#elif CONFIG_FRAMESIZE_SVGA
    int framesize = FRAMESIZE_SVGA;
    #define FRAMESIZE_STRING "800x600"
#elif CONFIG_FRAMESIZE_XGA
    int framesize = FRAMESIZE_XGA;
    #define FRAMESIZE_STRING "1024x768"
#elif CONFIG_FRAMESIZE_HD
    int framesize = FRAMESIZE_HD;
    #define FRAMESIZE_STRING "1280x720"
#elif CONFIG_FRAMESIZE_SXGA
    int framesize = FRAMESIZE_SXGA;
    #define FRAMESIZE_STRING "1280x1024"
#elif CONFIG_FRAMESIZE_UXGA
    int framesize = FRAMESIZE_UXGA;
    #define FRAMESIZE_STRING "1600x1200"
#endif

    ret = init_camera(framesize);
    if (ret != ESP_OK) {
        while(1) { vTaskDelay(1); }
    }

    REQUEST_t requestBuf;
    requestBuf.command = CMD_SEND;
    requestBuf.taskHandle = xTaskGetCurrentTaskHandle();
    snprintf(requestBuf.localFileName,
sizeof(requestBuf.localFileName)-1, "%s/picture.jpg", base_path);
    ESP_LOGI(TAG, "localFileName=%s",requestBuf.localFileName);

#if CONFIG_REMOTE_IS_FIXED_NAME
#if CONFIG_REMOTE_FRAMESIZE
    char baseFileName[32];
```

```
    strcpy(baseFileName, CONFIG_FIXED_REMOTE_FILE);
    for (int index=0;index<strlen(baseFileName);index++) {
        if (baseFileName[index] == 0x2E) baseFileName[index] = 0;
    }
    ESP_LOGI(TAG, "baseFileName=[%s]", baseFileName);
    sprintf(requestBuf.remoteFileName, "%s_%s.jpg", baseFileName,
FRAMESIZE_STRING);
#else
    sprintf(requestBuf.remoteFileName, "%s", CONFIG_FIXED_REMOTE_FILE);
#endif
    ESP_LOGI(TAG, "remoteFileName=%s",requestBuf.remoteFileName);
#endif

    HTTP_t httpBuf;
    httpBuf.taskHandle = xTaskGetCurrentTaskHandle();
    strcpy(httpBuf.localFileName, requestBuf.localFileName);
    CMD_t cmdBuf;

    while(1) {
        ESP_LOGI(TAG,"Waitting %s ....", SHUTTER);
        xQueueReceive(xQueueCmd, &cmdBuf, portMAX_DELAY);
        ESP_LOGI(TAG,"cmdBuf.command=%d", cmdBuf.command);
        if (cmdBuf.command == CMD_HALT) break;
        struct stat statBuf;
        if (stat(requestBuf.localFileName, &statBuf) == 0) {
            unlink(requestBuf.localFileName);
        }

#if CONFIG_REMOTE_IS_VARIABLE_NAME
        time(&now);
        now = now + (CONFIG_LOCAL_TIMEZONE*60*60);
        localtime_r(&now, &timeinfo);
        strftime(strftime_buf, sizeof(strftime_buf), "%c", &timeinfo);
        ESP_LOGI(TAG, "The current date/time is: %s", strftime_buf);
#if CONFIG_REMOTE_FRAMESIZE
        sprintf(requestBuf.remoteFileName,
"%04d%02d%02d-%02d%02d%02d_%s.jpg",
        (timeinfo.tm_year+1900),(timeinfo.tm_mon+1),timeinfo.tm_mday,
        timeinfo.tm_hour,timeinfo.tm_min,timeinfo.tm_sec,
FRAMESIZE_STRING);
#else
        sprintf(requestBuf.remoteFileName,
"%04d%02d%02d-%02d%02d%02d.jpg",
```

```
        (timeinfo.tm_year+1900),(timeinfo.tm_mon+1),timeinfo.tm_mday,
        timeinfo.tm_hour,timeinfo.tm_min,timeinfo.tm_sec);
#endif
        ESP_LOGI(TAG, "remoteFileName: %s", requestBuf.remoteFileName);
#endif

#if CONFIG_ENABLE_FLASH
        gpio_set_level(CONFIG_GPIO_FLASH, 1);
#endif

        int retryCounter = 0;
        while(1) {
            size_t pictureSize;
            ret = camera_capture(requestBuf.localFileName,
&pictureSize);
            ESP_LOGI(TAG, "camera_capture=%d",ret);
            ESP_LOGI(TAG, "pictureSize=%d",pictureSize);
            if (ret != ESP_OK) continue;
            struct stat statBuf;
            if (stat(requestBuf.localFileName, &statBuf) == 0) {
                ESP_LOGI(TAG, "st_size=%d", (int)statBuf.st_size);
                if (statBuf.st_size == pictureSize) break;
                retryCounter++;
                ESP_LOGI(TAG, "Retry capture %d",retryCounter);
                if (retryCounter > 10) {
                    ESP_LOGE(TAG, "Retry over for capture");
                    break;
                }
                vTaskDelay(1000);
            }
        }

#if CONFIG_ENABLE_FLASH
        gpio_set_level(CONFIG_GPIO_FLASH, 0);
#endif

        if (xQueueSend(xQueueRequest, &requestBuf, 10) != pdPASS) {
            ESP_LOGE(TAG, "xQueueSend fail");
        } else {
            uint32_t value = ulTaskNotifyTake(pdTRUE, portMAX_DELAY);
            ESP_LOGI(TAG, "ulTaskNotifyTake value=%"PRIx32, value);
        }
        if (xQueueSend(xQueueHttp, &httpBuf, 10) != pdPASS) {
```

```
            ESP_LOGE(TAG, "xQueueSend xQueueHttp fail");
        }
        vTaskDelay(1000/portTICK_PERIOD_MS);
    }
}
```