SISTEMA DE UBICACION PARA MASCOTAS

INDICE

Contenido

Sistema de ubicación para mascotas

1. Descripción General

1.1. Introducción y Funcionamiento general

El **Sistema de ubicación para mascotas,** desarrollado en la cátedra de Digitales II, de la Universidad Tecnológica Nacional de Buenos Aires, tiene la finalidad de monitorear a las mismas, controlando que no se muevan por fuera de un perímetro determinado.

En caso de que esta situación ocurriese, el sistema le proveerá periódicamente al usuario las coordenadas exactas de la posición de su mascota por medio de mensajes SMS.

El sistema esta implementado en base al microcontrolador LPC1769 de LPCXpresso.

1.2. Interfaz con el usuario

Todas las configuraciones necesarias para el correcto funcionamiento del sistema se pueden realizar por medio de mensajes SMS, setear el radio deseado para el perímetro de control, definir el punto de origen o sincronizar el tiempo señales de posición, son algunos de los ejemplos que se encuentran detallados en el manual del usuario

El equipo funciona con una alimentación de 3,3 y 1 mA que le provee un power bank, que esta capacitado para poder cargarse por via USB.

Se dispone de:

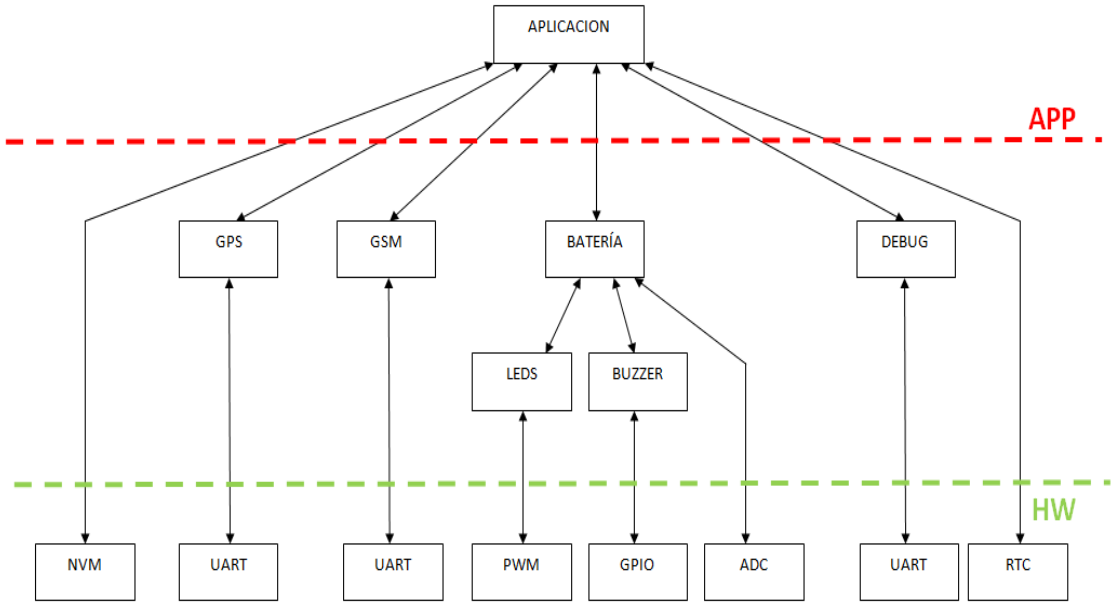
Un LED RGB para indicar el estado de recepción de señal GPS y GSM.

Un Switch manual para encendido del equipo.

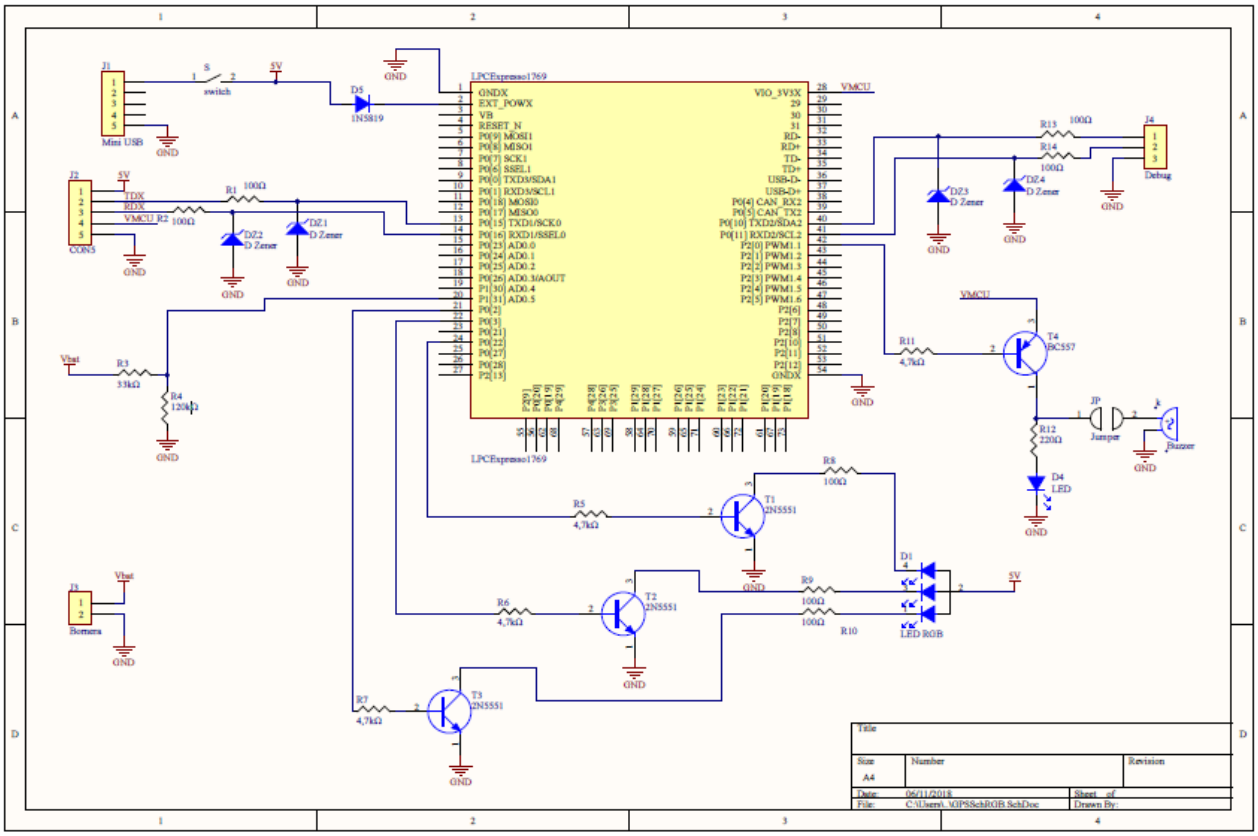
Un buzzer alertará al usuario en caso de batería baja.

2. Hardware

2.1. Diagrama en Bloques



2.2. Circuito esquemático



2.3. Descripción del Circuito y Hardware agregado

El proyecto está compuesto por las siguientes partes:

Microprocesador LPC1769, Basado en una arquitectura ARM Cortex-M3 para desarrollar aplicaciones integradas que ofrecen un alto nivel de integración y bajo consumo de energía.
El ARM Cortex-M3 CPU entre una de sus características incorpora tres “stage pipeline” y una arquitectura Harvard con instrucción de separación local.. Su elección fue decida en base a las características de bajo consumo, y su rendimiento a la velocidad de procesamiento para hacer una buena definición acorde a las necesidades del proyecto.

Circuito **LED RGB 5v ánodo común** conectado con **Transistores 2N5551** y resistencias smd

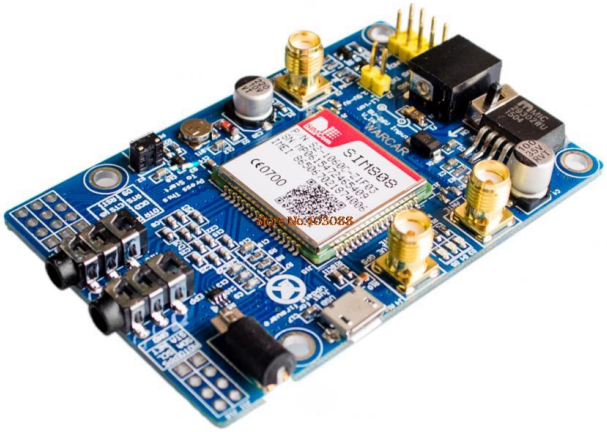
Circuito de debug con diodos zenner

Circuito de **Buzzer y LED** conectados con un transistor **BC557** y resistencias

Conector Mini USB junto con un switch y un diodo para protección **1N5819**

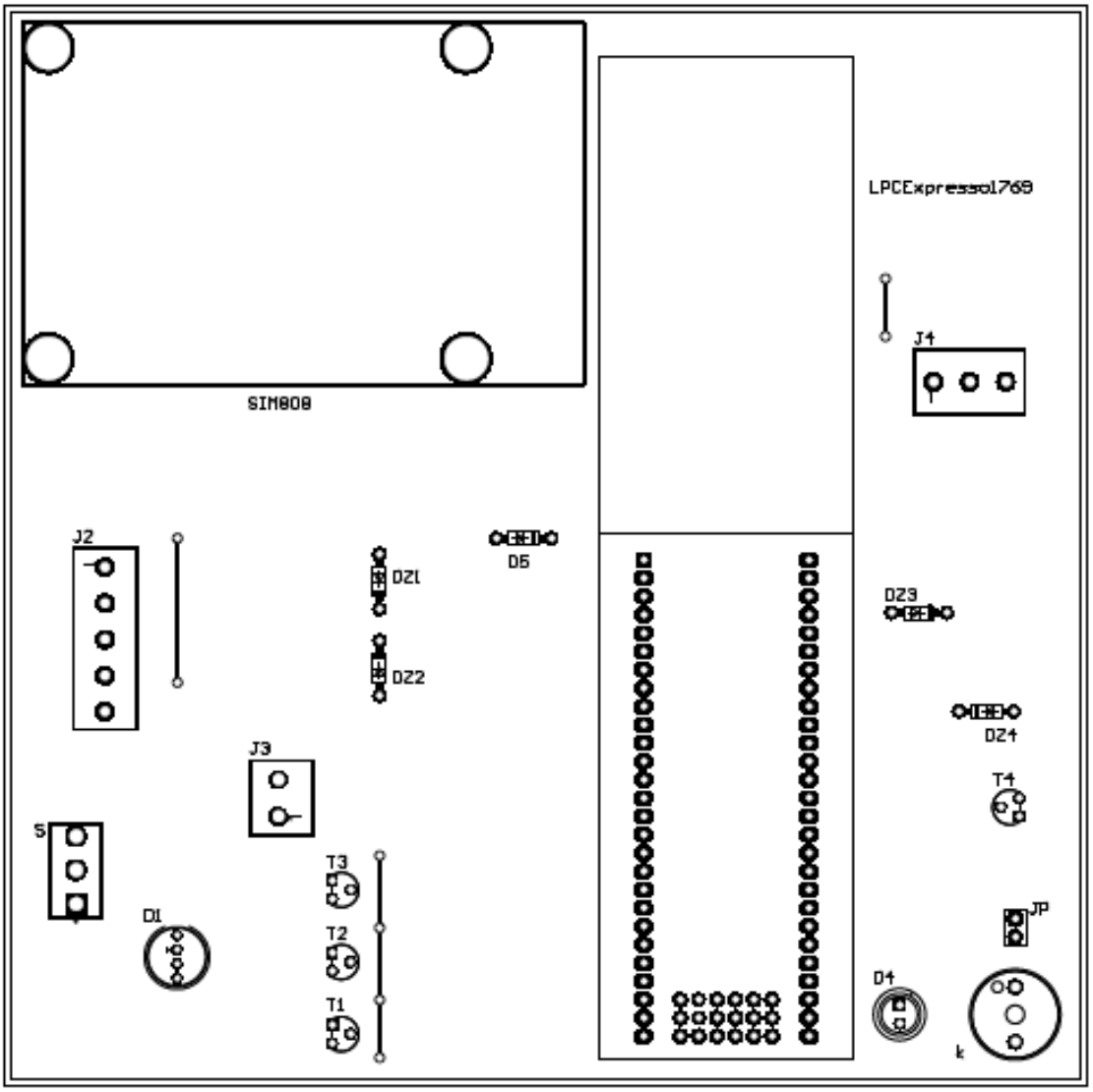
Borneras para alimentar y conectarse con el modulo GPS

Modulo GPS GSM SIM808
es un módulo con dos funciones principales. Su diseño se creó a partir del módulo GSM / GPS SIM808 de SIMCOM. Es compatible con GSM / GPRS de cuatro bandas. Combina la tecnología GPS para obtener la posición en latitud y longitud. Su diseño incorpora un modo de consumo de baja energía y puede conectarse con sistemas de energía a base de baterías de litio. Es compatible con A-GPS. El módulo se controla mediante comandos AT mediante una interfaz de comunicación serial, puede funcionar ya sea con una lógica de voltaje de 3.3V y/o 5V.

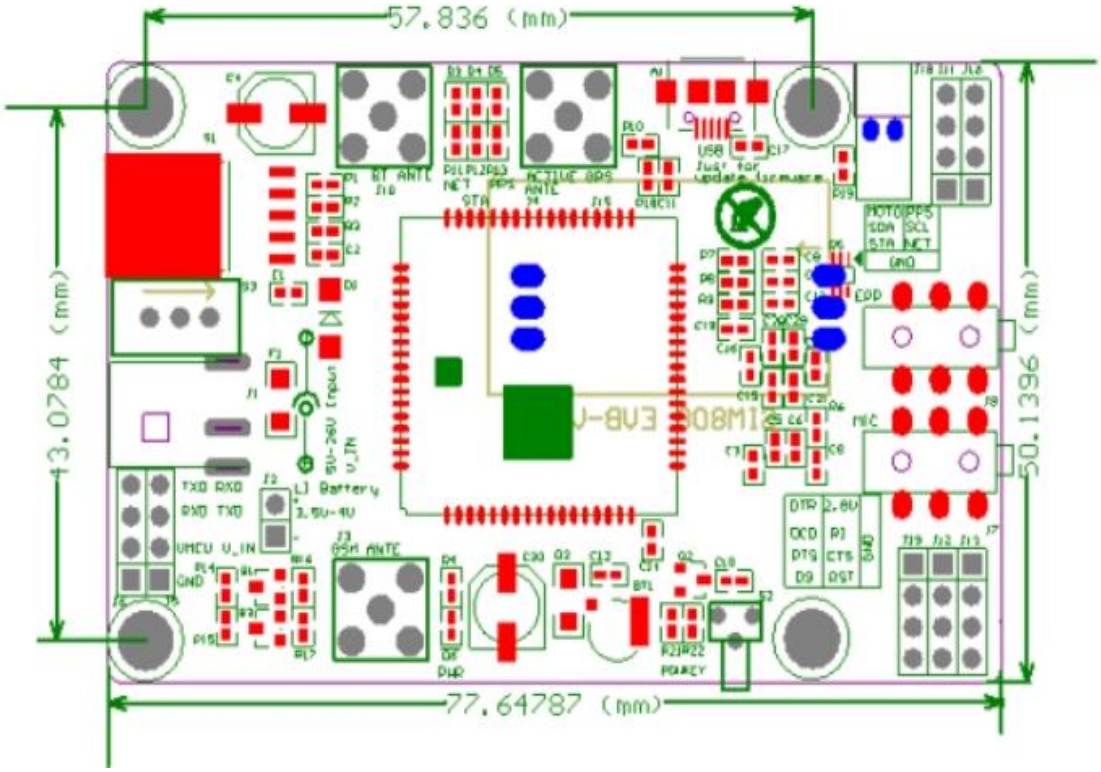


2.4. Circuito Impreso

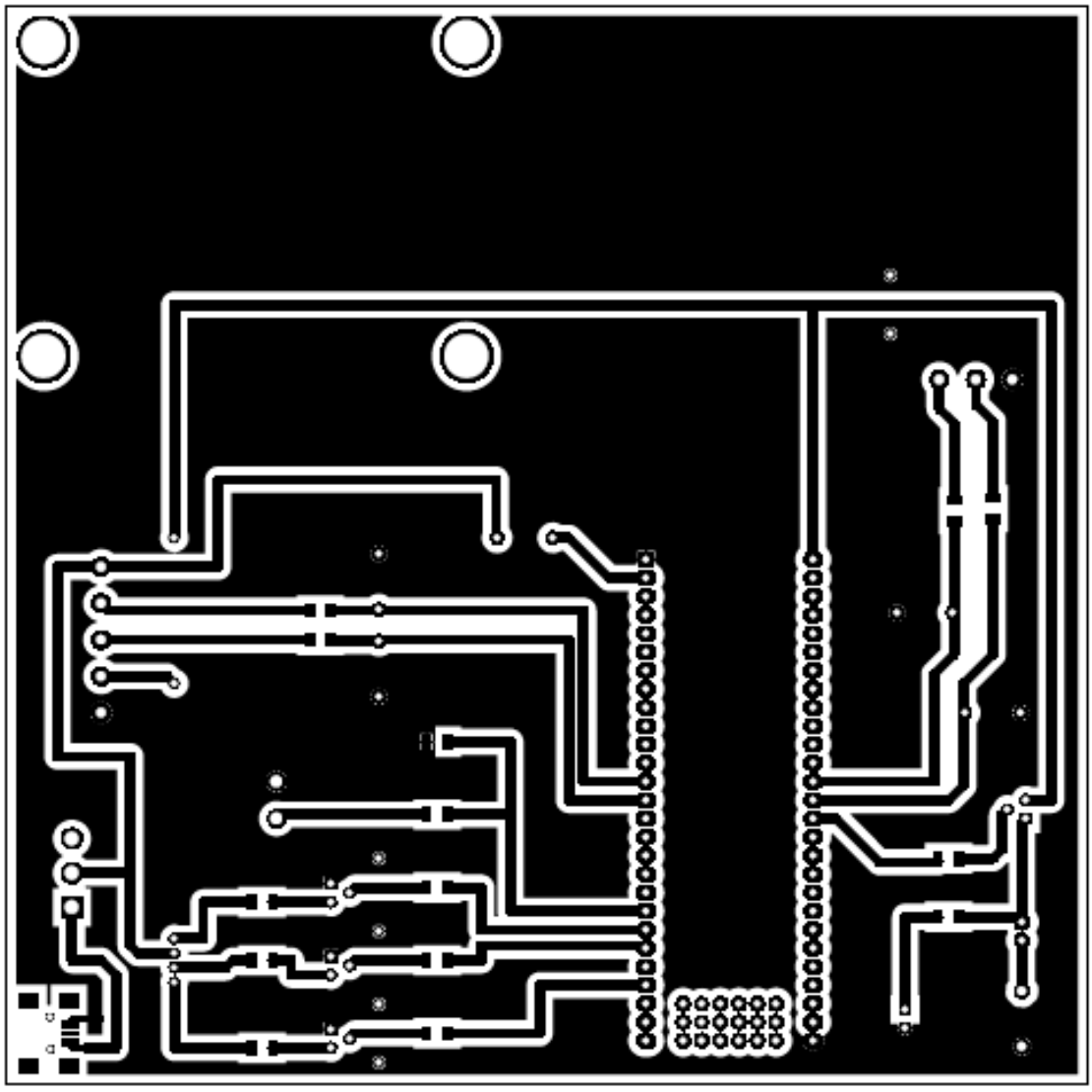
TOP LAYER



SIM 808



BOTTOM LAYER



3. Software

3.1. Software Microcontrolador

Utilizamos un microcontrolador Cortex M3 LPC1769.
Dicho dispositivo, cuenta con una gran variedad de recursos, de los cuales utilizamos:

UART: es la interfaz de comunicación con el módulo SIM808. Este módulo se utilizó para tener acceso a la redes GSM y GPS.

UART: utilizamos otra interfaz UART para conectar el dispositivo con la PC. Esto nos permitió obtener información en tiempo real del funcionamiento del equipo, como ser, estado de las conexiones GSM y GPS, latitud y longitud instantánea, estado de batería y otros datos menores

GPIOs: utilizamos varias GPIOs para comandar un led RGB y un Buzzer

3.1.1. Entorno de desarrollo

El entorno de desarrollo utilizado fue LPCXpresso. Todo el programa fue desarrollado en lenguaje C

3.1.2. Sistema Operativo

Utilizamos el sistema operativo FreeRTOS.
La utilización de O.S. nos facilitó la sincronización entre las tareas que componen al programa. Para esto, hicimos uso de varias queues, un mutex y un semáforo binario.

3.1.3. Librerías

El programa se construyó en base al repositorio "firmware" provisto por la cátedra. Sin embargo, las inicializaciones de los periféricos fueron realizadas por nosotros manualmente.

3.1.4. Programa Principal

El programa se compone de varias tareas, donde dos de ellas son primordiales. Una para manejar la interfaz GSM y otra para manejar la interfaz GPS.

3.1.5. Rutinas Generales

La tarea GPS se ocupa en primera instancia en verificar que el módulo tenga un nivel de señal aceptable, una vez que eso ocurre, comienza a recibir una nueva coordenada cada 3 segundos. Luego, decodifica los datos y encola la información en una queue.

La tarea GSM se ocupa en primera instancia en verificar que el dispositivo este conectado a un operador movil, cuando esto ocurre, verifica si hay mensajes pendientes para enviar y en caso que los haya, los envia.

Una tercera tarea, revisa constantemente la queue de coordenadas GPS y al recibir una nueva la compara contra las coordenadas de la zona segura definida por el usuario. Si la distancia es mayor al radio seguro, encola un mensaje SMS con las coordenadas actuales de la mascota.

Otras tareas de menor importancia, verifican cada 5 segundos el estado de batería y en caso que esté por debajo de un umbral definido en el programa, hace sonar el buzzer indicando una alerta.

4. Referencias

http://www.alldatasheet.com/
http://www.altium.com/.
https://www.genbeta.com/
https://acoptex.com/
https://www.google.com.ar/maps/
https://www.freertos.org/

5. Codigo

MAIN.c

```
/*********************************INCLUDES*********************************
**/
#include <string.h>
#include <stdlib.h>
#include "../inc/FreeRTOSConfig.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "semphr.h"

#include "../inc/PET_BOARD.h"
#include "../inc/RGB.h"
#include "../inc/BUZZER.h"
#include "../inc/UART.h"
#include "../inc/NVM.h"
#include "../inc/GPS.h"
#include "../inc/ADC.h"
/*************************************************************************
**/


/*********************************DATA
TYPES*********************************/
typedef struct response_tag
{
	uint8_t data[150];
	uint8_t size;
}response_t;
/*************************************************************************
**/


/*********************************DEFINES*********************************
**/
#define DISABLE_ECHO		0
#define WAIT_ECHO			1
#define	SMS_READY			2
#define	TEXT_MODE			3
#define	WAIT_TEXT_MODE		4
#define NETWORK_REG			5
#define	WAIT_NETWORK_REG	6
#define	READ_SMS			7
#define	WAIT_READ_SMS		8
#define DELETE_SMS			9
#define	WAIT_DELETE_SMS		10
#define SEND_SMS_1			11
#define WAIT_SEND_SMS_1		12
#define	SEND_SMS_2			13
#define	WAIT_SEND_SMS_2		14
#define NEXT				15

#define	WAIT_GSM_INIT		0
```

```
#define      POWER_ON                              1
#define WAIT_POWER_ON                2
#define CHECK_SIGNAL                 3
#define WAIT_CHECK_SIGNAL 4
#define      GET_COORDINATE          5
#define      WAIT_COORDINATE         6

#define      OFFLINE                                     0
#define      ONLINE                                1

#define      SZ_DATA_LOADED          0x12345678
/*************************************************************************
**/


/*********************************PROTOTYPES*********************************
**/
static void initHardware(void);
void task_rgb(void * a);
void task_buzzer(void * a);
static void task_adc(void * a);
static void task_gsm(void * a);
static void task_sim808_receive(void * a);
static void task_gps(void * a);
static void task_coordinates(void * a);
static void task_led(void * a);

static void store_phone_number(uint8_t * data);
static void store_sms_data(response_t response);
/*************************************************************************
**/


/*********************************VARIABLES**********************************
**/
const uint32_t OscRateIn = 12000000;
const uint32_t RTCOscRateIn = 32768;

double sz_latitude;
double sz_longitude;
uint32_t sz_radius;
uint8_t phone_number[14];
uint32_t device_status;

uint8_t sms_check;
uint8_t gps_status;
uint8_t gsm_status;
uint8_t sms_hold;

xQueueHandle queue_rgb;
xQueueHandle queue_buzzer;
xQueueHandle queue_uart_0_tx;
xQueueHandle queue_uart_0_rx;
xQueueHandle queue_uart_1_tx;
xQueueHandle queue_uart_1_rx;
xQueueHandle queue_uart_2_tx;
xQueueHandle queue_uart_2_rx;
xQueueHandle queue_adc;
xQueueHandle queue_sim808;
```

```
xQueueHandle queue_coordinates;
xQueueHandle queue_sms;

xSemaphoreHandle mutex_sim808;
xSemaphoreHandle binary_sim808;
/*****************************************************************************
**/

int main(void)
{
	queue_rgb = xQueueCreate(1, sizeof(uint8_t));
	queue_buzzer = xQueueCreate(1, sizeof(uint8_t));
	queue_uart_0_tx = xQueueCreate(100, sizeof(uint8_t));
	queue_uart_0_rx = xQueueCreate(100, sizeof(uint8_t));
	queue_uart_1_tx = xQueueCreate(100, sizeof(uint8_t));
	queue_uart_1_rx = xQueueCreate(100, sizeof(uint8_t));
	queue_uart_2_tx = xQueueCreate(100, sizeof(uint8_t));
	queue_uart_2_rx = xQueueCreate(100, sizeof(uint8_t));
	queue_adc = xQueueCreate(1, sizeof(uint16_t));
	queue_sim808 = xQueueCreate(5, sizeof(response_t));
	queue_coordinates = xQueueCreate(1, sizeof(coordinate_t));
	queue_sms = xQueueCreate(1, sizeof(coordinate_t));

	mutex_sim808 = xSemaphoreCreateMutex();
	binary_sim808 = xSemaphoreCreateCounting(1,0);

	initHardware();

	xTaskCreate(task_rgb, (const char *)"task_rgb",
configMINIMAL_STACK_SIZE*2, 0, tskIDLE_PRIORITY+1, 0);
	xTaskCreate(task_buzzer, (const char *)"task_buzzer",
configMINIMAL_STACK_SIZE*2, 0, tskIDLE_PRIORITY+1, 0);
	xTaskCreate(task_adc, (const char *)"task_adc",
configMINIMAL_STACK_SIZE*2, 0, tskIDLE_PRIORITY+1, 0);
	xTaskCreate(task_gsm, (const char *)"task_gsm",
configMINIMAL_STACK_SIZE*2, 0, tskIDLE_PRIORITY+1, 0);
	xTaskCreate(task_sim808_receive, (const char *)"task_sim808_receive",
configMINIMAL_STACK_SIZE*2, 0, tskIDLE_PRIORITY+1, 0);
	xTaskCreate(task_gps, (const char *)"task_gps",
configMINIMAL_STACK_SIZE*2, 0, tskIDLE_PRIORITY+1, 0);
	xTaskCreate(task_coordinates, (const char *)"task_coordinates",
configMINIMAL_STACK_SIZE*2, 0, tskIDLE_PRIORITY+1, 0);
	xTaskCreate(task_led, (const char *)"task_led",
configMINIMAL_STACK_SIZE*2, 0, tskIDLE_PRIORITY+1, 0);

	vTaskStartScheduler();

	while (1)
	{
	}
}


static void initHardware(void)
{
	rgb_config_t rgb_cfg;
	buzzer_config_t buzzer_cfg;
	uart_config_t uart_cfg;
```

```
	SystemCoreClockUpdate();

	nvm_init();

	rgb_cfg.red_port = RGB_RED_PORT;
	rgb_cfg.red_pin = RGB_RED_PIN;
	rgb_cfg.green_port = RGB_GREEN_PORT;
	rgb_cfg.green_pin = RGB_GREEN_PIN;
	rgb_cfg.blue_port = RGB_BLUE_PORT;
	rgb_cfg.blue_pin = RGB_BLUE_PIN;
	rgb_init(&rgb_cfg);

	buzzer_cfg.port = BUZZER_PORT;
	buzzer_cfg.pin = BUZZER_PIN;
	buzzer_init(&buzzer_cfg);

	uart_cfg.uart_number=UART_DEBUG;
	uart_cfg.baudrate=9600;
	uart_init(&uart_cfg);

	uart_cfg.uart_number=UART_COMM;
	uart_cfg.baudrate=9600;
	uart_init(&uart_cfg);

	adc_init();

	rgb_set(OFF);
	buzzer_set(BUZZER_OFF);
}


static void task_coordinates(void * a)
{
	static coordinate_t new_coordinate;
	static double distance;
	static char aux[50];

	while (1)
	{
		xQueueReceive(queue_coordinates, &new_coordinate,
portMAX_DELAY);

		if (device_status == SZ_DATA_LOADED)
		{
			sprintf(aux, "Latitud: %lf\n", new_coordinate.latitude);
			uart_send_data(UART_DEBUG, (uint8_t*)aux, strlen(aux));

			sprintf(aux, "Longitud: %lf\n", new_coordinate.longitude);
			uart_send_data(UART_DEBUG, (uint8_t*)aux, strlen(aux));

			distance = calculate_distance(sz_latitude, sz_longitude,
new_coordinate.latitude, new_coordinate.longitude);

			sprintf(aux, "Distancia: %lf\n\n", distance);
			uart_send_data(UART_DEBUG, (uint8_t*)aux, strlen(aux));

			if (distance > sz_radius && gsm_status == ONLINE &&
sms_hold == 0)
			{
```

```
                        sms_hold=1;
                        xQueueSend(queue_sms, &new_coordinate, 0);
                }
        }
}


static void task_gsm(void * a)
{
        static uint8_t status=0;
        static response_t last_response;
        static coordinate_t coord;
        static char aux[100];

        while (1)
        {
                switch(status)
                {
                        case DISABLE_ECHO:
                                vTaskDelay(10 / portTICK_RATE_MS);
                                xQueueReset(queue_sim808);
                                uart_send_data(UART_COMM, (uint8_t*)"ATE0\r", 5);
                                status = WAIT_ECHO;
                                break;

                        case WAIT_ECHO:
                                if (xQueueReceive(queue_sim808, &last_response,
5000) == pdTRUE)
                                {
                                        if (!memcmp(last_response.data, "OK", 2))
                                        {
                                                uart_send_data(UART_DEBUG,
(uint8_t*)"Echo Deshabilitado!\n", strlen("Echo Deshabilitado!\n"));
                                                status = SMS_READY;
                                        }
                                }
                                else
                                {
                                        status = DISABLE_ECHO;
                                }
                                break;

                        case SMS_READY:
                                xQueueReceive(queue_sim808, &last_response,
portMAX_DELAY);

                                if (!memcmp(last_response.data, "SMS Ready", 9))
                                {
                                        uart_send_data(UART_DEBUG, (uint8_t*)"SMS
Ready Recibido!\n", strlen("SMS Ready Recibido!\n"));
                                        status = TEXT_MODE;
                                        xSemaphoreGive(binary_sim808);
                                }
                                break;

                        case TEXT_MODE:
                                xSemaphoreTake(mutex_sim808, portMAX_DELAY);
                                vTaskDelay(10 / portTICK_RATE_MS);
```

```
                        xQueueReset(queue_sim808);
                        uart_send_data(UART_COMM, (uint8_t*)"AT+CMGF=1\r",
strlen("AT+CMGF=1\r"));
                        status = WAIT_TEXT_MODE;
                        break;

                case WAIT_TEXT_MODE:
                        if (xQueueReceive(queue_sim808, &last_response,
5000) == pdTRUE)
                        {
                                if (!memcmp(last_response.data, "OK", 2))
                                {
                                        uart_send_data(UART_DEBUG,
(uint8_t*)"Configurado Modo Texto!\n", strlen("Configurado Modo Texto!\n"));
                                        status = NETWORK_REG;
                                }
                                else
                                {
                                        status = TEXT_MODE;
                                }
                        }
                        else
                        {
                                status = TEXT_MODE;
                        }
                        xSemaphoreGive(mutex_sim808);
                        break;

                case NETWORK_REG:
                        xSemaphoreTake(mutex_sim808, portMAX_DELAY);
                        vTaskDelay(10 / portTICK_RATE_MS);
                        xQueueReset(queue_sim808);
                        uart_send_data(UART_COMM, (uint8_t*)"AT+CREG?\r",
strlen("AT+CREG?\r"));
                        status = WAIT_NETWORK_REG;
                        break;

                case WAIT_NETWORK_REG:
                        if (xQueueReceive(queue_sim808, &last_response,
5000) == pdTRUE)
                        {
                                uart_send_data(UART_DEBUG, (uint8_t*)"RTA: ",
5);
                                uart_send_data(UART_DEBUG,
last_response.data, last_response.size);
                                uart_send_data(UART_DEBUG, (uint8_t*)"\n",
1);

                                if (!memcmp(last_response.data, "+CREG: 0,1",
10))
                                {
                                        gsm_status=ONLINE;
                                        uart_send_data(UART_DEBUG,
(uint8_t*)"Registrado a la Red!\n", strlen("Registrado a la Red!\n"));
                                        status = READ_SMS;
                                }
                                else
                                {
                                        vTaskDelay(5000 / portTICK_RATE_MS);
```

```
                                        status = NETWORK_REG;
                                }
                        }
                        else
                        {
                                vTaskDelay(5000 / portTICK_RATE_MS);
                                status = NETWORK_REG;
                        }
                        xSemaphoreGive(mutex_sim808);
                        break;

                case READ_SMS:
                        xSemaphoreTake(mutex_sim808, portMAX_DELAY);
                        vTaskDelay(10 / portTICK_RATE_MS);
                        xQueueReset(queue_sim808);
                        sms_check=1;
                        uart_send_data(UART_COMM,
(uint8_t*)"AT+CMGL=\"ALL\"\r", strlen("AT+CMGL=\"ALL\"\r"));
                        status = WAIT_READ_SMS;
                        break;

                case WAIT_READ_SMS:
                        if (xQueueReceive(queue_sim808, &last_response,
5000) == pdTRUE)
                        {
                                if (!memcmp(last_response.data, "+CMGL: 1",
strlen("+CMGL: 1")))
                                {

      store_phone_number(last_response.data);

                                        xQueueReceive(queue_sim808,
&last_response, portMAX_DELAY);
                                        store_sms_data(last_response);
                                        status = DELETE_SMS;
                                }
                                else
                                {
                                        status = SEND_SMS_1;
                                }
                        }
                        else
                        {
                                status = READ_SMS;
                        }
                        xSemaphoreGive(mutex_sim808);
                        break;

                case DELETE_SMS:
                        xSemaphoreTake(mutex_sim808, portMAX_DELAY);
                        vTaskDelay(10 / portTICK_RATE_MS);
                        xQueueReset(queue_sim808);
                        uart_send_data(UART_COMM, (uint8_t*)"AT+CMGDA=\"DEL
ALL\"\r", strlen("AT+CMGDA=\"DEL ALL\"\r"));
                        status = WAIT_DELETE_SMS;
                        break;

                case WAIT_DELETE_SMS:
```

```
                        if (xQueueReceive(queue_sim808, &last_response,
5000) == pdTRUE)
                        {
                                if (!memcmp(last_response.data, "OK", 2))
                                {
                                        uart_send_data(UART_DEBUG,
(uint8_t*)"SMS Borrado!\n\n", strlen("SMS Borrado!\n\n"));
                                        status = SEND_SMS_1;
                                }
                                else
                                {
                                        status = DELETE_SMS;
                                }
                        }
                        else
                        {
                                status = DELETE_SMS;
                        }
                        xSemaphoreGive(mutex_sim808);
                        break;

                case SEND_SMS_1:
                        if (xQueueReceive(queue_sms, &coord, 0) == pdTRUE)
                        {

                                xSemaphoreTake(mutex_sim808, portMAX_DELAY);
                                vTaskDelay(10 / portTICK_RATE_MS);
                                xQueueReset(queue_sim808);
                                sprintf(aux,
"AT+CMGS=\"%s\"\r",phone_number);
                                uart_send_data(UART_COMM, (uint8_t*)aux,
strlen(aux));
                                status = WAIT_SEND_SMS_1;


                                /*
                                uart_send_data(UART_DEBUG,
(uint8_t*)"MANDA!\n\n", strlen("MANDA!\n\n"));
                                status=NEXT;
                                */
                        }
                        else
                        {
                                vTaskDelay(3000 / portTICK_RATE_MS);
                                status = READ_SMS;
                        }
                        break;

                case WAIT_SEND_SMS_1:
                        if (xQueueReceive(queue_sim808, &last_response,
10000) == pdTRUE)
                        {
                                if (!memcmp(last_response.data, ">", 1))
                                {
                                        status = SEND_SMS_2;
                                }
                        }
                        xSemaphoreGive(mutex_sim808);
                        break;
```

```
                    case SEND_SMS_2:
                          xSemaphoreTake(mutex_sim808, portMAX_DELAY);
                          vTaskDelay(10 / portTICK_RATE_MS);
                          xQueueReset(queue_sim808);
                          sprintf(aux,"Ubicacion mascota:\n\nLatitud:
%lf\nLongitud: %lf\x1a", coord.latitude, coord.longitude);
                          uart_send_data(UART_COMM, (uint8_t*)aux,
strlen(aux));
                          status = WAIT_SEND_SMS_2;
                          break;

                    case WAIT_SEND_SMS_2:
                          if (xQueueReceive(queue_sim808, &last_response,
60000) == pdTRUE)
                          {
                                if (!memcmp(last_response.data, "+CMGS:", 6))
                                {
                                      uart_send_data(UART_DEBUG,
(uint8_t*)"SMS Enviado!\n\n", strlen("SMS Enviado!\n\n"));
                                      status = NEXT;
                                }
                                else
                                {
                                      uart_send_data(UART_DEBUG,
(uint8_t*)"SMS ERROR\n\n", strlen("SMS ERROR\n\n"));
                                      status = NEXT;
                                }
                          }
                          else
                          {
                                uart_send_data(UART_DEBUG, (uint8_t*)"SMS
TIMEOUT!\n\n", strlen("SMS TIMEOUT!\n\n"));
                                status = NEXT;
                          }
                          xSemaphoreGive(mutex_sim808);
                          break;

                    case NEXT:
                          vTaskDelay(60000 / portTICK_RATE_MS);
                          sms_hold=0;
                          status = READ_SMS;
                          break;
             }
      }
}



static void task_gps(void * a)
{
      static uint8_t status=0;
      static response_t last_response;

      while(1)
      {
            switch(status)
            {
                    case WAIT_GSM_INIT:
```

```
                        xSemaphoreTake(binary_sim808, portMAX_DELAY);
                        status = POWER_ON;
                        break;

                case POWER_ON:
                        xSemaphoreTake(mutex_sim808, portMAX_DELAY);
                        vTaskDelay(10 / portTICK_RATE_MS);
                        xQueueReset(queue_sim808);
                        uart_send_data(UART_COMM,
(uint8_t*)"AT+CGPSPWR=1\r", strlen("AT+CGPSPWR=1\r"));
                        status = WAIT_POWER_ON;
                        break;

                case WAIT_POWER_ON:
                        if (xQueueReceive(queue_sim808, &last_response,
5000) == pdTRUE)
                        {
                                if (!memcmp(last_response.data, "OK", 2))
                                {
                                        uart_send_data(UART_DEBUG,
(uint8_t*)"GPS Encendido!\n", strlen("GPS Encendido!\n"));
                                        status = CHECK_SIGNAL;
                                }
                                else
                                {
                                        status = POWER_ON;
                                }
                        }
                        else
                        {
                                status = POWER_ON;
                        }
                        xSemaphoreGive(mutex_sim808);
                        break;

                case CHECK_SIGNAL:
                        xSemaphoreTake(mutex_sim808, portMAX_DELAY);
                        vTaskDelay(100 / portTICK_RATE_MS);
                        xQueueReset(queue_sim808);
                        uart_send_data(UART_COMM,
(uint8_t*)"AT+CGPSSTATUS?\r", strlen("AT+CGPSSTATUS?\r"));
                        status = WAIT_CHECK_SIGNAL;
                        break;

                case WAIT_CHECK_SIGNAL:
                        if (xQueueReceive(queue_sim808, &last_response,
5000) == pdTRUE)
                        {
                                uart_send_data(UART_DEBUG, (uint8_t*)"RTA: ",
5);
                                uart_send_data(UART_DEBUG,
last_response.data, last_response.size);
                                uart_send_data(UART_DEBUG, (uint8_t*)"\n",
1);

                                if (!memcmp(last_response.data, "+CGPSSTATUS:
Location 3D Fix", strlen("+CGPSSTATUS: Location 3D Fix")))
                                {
```

```
                                        uart_send_data(UART_DEBUG,
(uint8_t*)"GPS Obtenido!\n\n", strlen("GPS Obtenido!\n\n"));
                                        gps_status = ONLINE;
                                        status = GET_COORDINATE;
                                }
                                else
                                {
                                        status = CHECK_SIGNAL;
                                }
                        }
                        else
                        {
                                status = CHECK_SIGNAL;
                        }
                        xSemaphoreGive(mutex_sim808);
                        vTaskDelay(3000 / portTICK_RATE_MS);
                        break;

                case GET_COORDINATE:
                        xSemaphoreTake(mutex_sim808, portMAX_DELAY);
                        vTaskDelay(10 / portTICK_RATE_MS);
                        xQueueReset(queue_sim808);
                        uart_send_data(UART_COMM,
(uint8_t*)"AT+CGPSINF=0\r", strlen("AT+CGPSINF=0\r"));
                        status = WAIT_COORDINATE;
                        break;

                case WAIT_COORDINATE:
                        if (xQueueReceive(queue_sim808, &last_response,
5000) == pdTRUE)
                        {
                                if (!memcmp(last_response.data, "+CGPSINF:",
strlen("+CGPSINF:")))
                                {
                                        if (parse_frame(last_response.data,
last_response.size))
                                        {
                                                status = GET_COORDINATE;
                                        }
                                        else
                                        {
                                                gps_status = OFFLINE;
                                                status = CHECK_SIGNAL;
                                        }
                                }
                                else
                                {
                                        status = GET_COORDINATE;
                                }
                        }
                        else
                        {
                                status = GET_COORDINATE;
                        }
                        xSemaphoreGive(mutex_sim808);
                        vTaskDelay(3000 / portTICK_RATE_MS);
                        break;
                }
        }
```

```
}


static void task_sim808_receive(void * a)
{
	static uint8_t data;
	static uint8_t index=0;
	static uint8_t status=0;
	static response_t last_response;

	while(1)
	{
		xQueueReceive(queue_uart_1_rx, &data, portMAX_DELAY);

		switch(status)
		{
			case 0:
				if (data == 0x0D)
				{
					status=1;
				}
				break;

			case 1:
				if (data == 0x0A)
				{
					status=2;
					last_response.size=0;
				}
				break;

			case 2:
				if (data != 0x0D)
				{
					last_response.data[index++]=data;
					last_response.size++;

					if (data == '>')
					{
						xQueueSend(queue_sim808,
&last_response, portMAX_DELAY);
						index=0;
						status=0;
					}
				}
				else
				{
					xQueueSend(queue_sim808, &last_response,
portMAX_DELAY);
					status=3;
				}
				break;

			case 3:
				if (data == 0x0A)
				{
					index=0;

					if (sms_check)
```

```
                                {
                                        if (last_response.size != 2)
                                        {
                                                status=2;
                                                last_response.size=0;
                                        }
                                        else
                                        {
                                                status=0;
                                        }
                                }
                                else
                                {
                                        status=0;
                                }

                                sms_check=0;
                        }
                        break;
                }
        }
}


static void task_adc(void * a)
{
        static uint16_t data;
        static char aux[50];
        static double counts;
        static double tension;

        while (1)
        {
                adc_soc();

                xQueueReceive(queue_adc, &data, portMAX_DELAY);
                counts = data;
                tension = counts * 0.000967441860;
                sprintf(aux, "Tension: %.2lf V\n", tension);
                uart_send_data(UART_DEBUG, (uint8_t*)aux, strlen(aux));

                if (tension < 2)
                {
                        buzzer_set(LOW_BAT);
                }

                vTaskDelay(5000 / portTICK_RATE_MS);
        }
}


static void store_phone_number(uint8_t * data)
{
        static uint8_t index=0;
        static uint8_t i=0;
        static uint32_t phone_number_nvm;

        /* index=1 para que no capture el '+' inicial */
        index=1;
```

```
	while(data[index] != '+')
	{
		index++;
	}

	for(i=0 ; i<13 ; i++)
	{
		phone_number[i]=data[index];
		index++;
	}

	phone_number[i]='\0';

	uart_send_data(UART_DEBUG, (uint8_t*)"\n", 1);
	uart_send_data(UART_DEBUG, (uint8_t*)"Numero: ", strlen("Numero: "));
	uart_send_data(UART_DEBUG, phone_number, strlen((char*)phone_number));
	uart_send_data(UART_DEBUG, (uint8_t*)"\n", 1);

	phone_number_nvm = strtol((char*)(phone_number + 3), NULL, 10);
	nvm_set(NUMBER, phone_number_nvm);
}


static void store_sms_data(response_t response)
{
	static uint8_t i=0;
	static uint8_t index=0;
	static uint8_t commas=0;
	static uint8_t latitude_string[20];
	static uint8_t longitude_string[20];
	static uint8_t radius_string[20];
	static char aux[50];
	static double aux_double;
	static uint32_t aux_reg;

	while(i < response.size)
	{
		if (response.data[i]==',')
		{
			commas++;
		}

		i++;
	}

	if (commas==2)
	{
		i=0;

		while(response.data[i] != ',')
		{
			latitude_string[index]=response.data[i];
			index++;
			i++;
		}

		latitude_string[index]='\0';
		index=0;
```

```
		i++;

		while(response.data[i] != ',')
		{
			longitude_string[index]=response.data[i];
			index++;
			i++;
		}

		longitude_string[index]='\0';
		index=0;
		i++;

		while(i < response.size)
		{
			radius_string[index]=response.data[i];
			index++;
			i++;
		}

		radius_string[index]='\0';
	}

	i=0;
	index=0;
	commas=0;

	sz_radius = strtol((char*)radius_string, NULL, 10);
	nvm_set(RADIUS, sz_radius);

	sprintf(aux, "Radio: %lu\n", sz_radius);
	uart_send_data(UART_DEBUG, (uint8_t*)aux, strlen(aux));

	aux_double = strtod((char*)latitude_string, NULL);
	sz_latitude = aux_double;

	if(aux_double < 0)
	{
		aux_double *= (-1);
	}

	aux_double *= 10000;
	aux_reg = (uint32_t) aux_double;
	nvm_set(LATITUDE, aux_reg);

	sprintf(aux, "Latitud SZ: %lf\n", sz_latitude);
	uart_send_data(UART_DEBUG, (uint8_t*)aux, strlen(aux));

	aux_double = strtod((char*)longitude_string, NULL);
	sz_longitude = aux_double;

	if(aux_double < 0)
	{
		aux_double *= (-1);
	}

	aux_double *= 10000;
	aux_reg = (uint32_t) aux_double;
	nvm_set(LONGITUDE, aux_reg);
```

```
	sprintf(aux, "Longitud SZ: %lf\n", sz_longitude);
	uart_send_data(UART_DEBUG, (uint8_t*)aux, strlen(aux));

	device_status = SZ_DATA_LOADED;
	nvm_set(STATUS, device_status);
}



static void task_led(void * a)
{
	while (1)
	{
		vTaskDelay(1000 / portTICK_RATE_MS);

		if (gps_status == OFFLINE && gsm_status == OFFLINE)
		{
			rgb_set(RED);
		}
		else if (gps_status == OFFLINE)
		{
			rgb_set(BLUE);
		}
		else if (gsm_status == OFFLINE)
		{
			rgb_set(YELLOW);
		}
		else if (device_status != SZ_DATA_LOADED)
		{
			rgb_set(MAGENTA);
		}
		else
		{
			rgb_set(GREEN);
		}
	}
}
```

GPS.c

```
/*********************************INCLUDES*********************************
**/
#include <stdint.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include "../inc/UART.h"

#include "../inc/FreeRTOSConfig.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
/*****************************************************************************
**/


/*********************************DEFINES*********************************
**/
#define d2r (M_PI / 180.0)
/*****************************************************************************
**/


/*********************************DATA
TYPES*********************************/
typedef struct coordinate_tag
{
	uint8_t latitude_string[20];
	uint8_t longitude_string[20];
	double latitude;
	double longitude;
}coordinate_t;
/*****************************************************************************
**/


/*********************************PROTOTYPES*********************************
**/
uint8_t parse_frame(uint8_t * frame, uint8_t size);
double nmea_deg2double(double deg);
double calculate_distance(double lat1, double lon1, double lat2, double
lon2);
/*****************************************************************************
**/


/*********************************VARIABLES*********************************
**/
extern xQueueHandle queue_coordinates;
/*****************************************************************************
**/

uint8_t parse_frame(uint8_t * frame, uint8_t size)
{
	static uint8_t index=0;
```

```
static uint8_t commas=0;
coordinate_t new_coordinate;
static uint8_t coord_index=0;
static uint8_t satellites_string[5];
static uint32_t satellites_int=0;
uint8_t ret=0;
static double aux_double;

while(index < size)
{
	if (frame[index] == ',')
	{
		commas++;
	}

	index++;
}

index=0;

if (commas == 8)
{
	while(frame[index] != ',')
	{
		index++;
	}

	index++;

	while(frame[index] != ',')
	{
		new_coordinate.latitude_string[coord_index]=frame[index];
		index++;
		coord_index++;
	}

	new_coordinate.latitude_string[coord_index]='\0';
	coord_index=0;
	index++;

	while(frame[index] != ',')
	{
		new_coordinate.longitude_string[coord_index]=frame[index];
		index++;
		coord_index++;
	}

	new_coordinate.longitude_string[coord_index]='\0';
	coord_index=0;
	index++;

	while(frame[index] != ',')
	{
		index++;
	}

	index++;

	while(frame[index] != ',')
```

```
		{
			index++;
		}

		index++;

		while(frame[index] != ',')
		{
			index++;
		}

		index++;

		while(frame[index] != ',')
		{
			satellites_string[coord_index]=frame[index];
			index++;
			coord_index++;
		}

		satellites_string[coord_index]='\0';
		coord_index=0;
		index++;

		satellites_int = strtol((char*)satellites_string, NULL, 10);

		if (satellites_int >= 4)
		{
			aux_double = strtod((char*)new_coordinate.latitude_string,
NULL);
			aux_double *= (-1);
			new_coordinate.latitude = nmea_deg2double(aux_double);

			aux_double =
strtod((char*)new_coordinate.longitude_string, NULL);
			aux_double *= (-1);
			new_coordinate.longitude = nmea_deg2double(aux_double);

			xQueueSend(queue_coordinates, &new_coordinate,
portMAX_DELAY);
			ret=1;
		}
		else
		{
			ret=0;
		}
	}

	index=0;
	commas=0;
	coord_index=0;

	return ret;
}


double calculate_distance(double lat1, double lon1, double lat2, double lon2)
{
	static double pi80;
```

```
	static double r;
	static double dlat;
	static double dlon;
	static double a;
	static double c;
	static double z;
	static double ret;


	pi80 = M_PI / 180;
	lat1 *= pi80;
	lat2 *= pi80;
	lon1 *= pi80;
	lon2 *= pi80;

	r = 6372.797;
	dlat = lat2 - lat1;
	dlon = lon2 - lon1;

	a = sin(dlat/2) * sin(dlat/2) + cos(lat1) * cos(lat2) * sin(dlon/2) *
sin(dlon/2);
	c = 2 * atan2(sqrt(a),sqrt(1-a));

	z = r*c;
	ret = z*1000;

	return ret;
	//return (r*c)*1000;
}



double nmea_deg2double(double deg)
{
	static double ret;
	static double decimales;

	ret = (int) deg/100 ;
	decimales = deg - (ret*100);
	decimales /= 60;
	ret += decimales;

	return ret;
}
```

GPS.h

```
/*********************************INCLUDES*********************************
**/
#include <stdint.h>
/*************************************************************************
**/


/*********************************DATA
TYPES*********************************/
typedef struct coordinate_tag
{
	uint8_t latitude_string[20];
	uint8_t longitude_string[20];
	double latitude;
	double longitude;
}coordinate_t;
/*************************************************************************
**/


/*********************************PROTOTYPES*******************************
**/
uint8_t parse_frame(uint8_t * frame, uint8_t size);
double calculate_distance(double lat1, double lon1, double lat2, double
lon2);
/*************************************************************************
**/
```

UART.c

```
/*********************************INCLUDES*********************************
**/
#include <stdint.h>
#include "../inc/GPIO.h"

#include "../inc/FreeRTOSConfig.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
/*************************************************************************
**/


/*********************************DEFINES**********************************
**/
#define PCONP                   (*((volatile uint32_t *) 0x400FC0C4UL))
#define      PCLKSEL            ((volatile uint32_t *) 0x400FC1A8UL)
#define      PCLKSEL0     PCLKSEL[0]
#define      PCLKSEL1     PCLKSEL[1]

#define      ISER               ((volatile uint32_t *) 0xE000E100UL)
#define      ISER0              ISER[0]

#define      DIR_UART0    ((volatile uint32_t *) 0x4000C000UL)
#define      U0RBR              DIR_UART0[0]
#define      U0THR              DIR_UART0[0]
#define      U0DLL              DIR_UART0[0]
#define      U0IER              DIR_UART0[1]
#define      U0DLM              DIR_UART0[1]
#define      U0IIR              DIR_UART0[2]
#define      U0LCR              DIR_UART0[3]

#define DIR_UART1   ((volatile uint32_t *) 0x40010000UL)
#define      U1RBR              DIR_UART1[0]
#define      U1THR              DIR_UART1[0]
#define      U1DLL              DIR_UART1[0]
#define      U1IER              DIR_UART1[1]
#define      U1DLM              DIR_UART1[1]
#define      U1IIR              DIR_UART1[2]
#define      U1LCR              DIR_UART1[3]

#define      DIR_UART2    ((volatile uint32_t *) 0x40098000UL)
#define      U2RBR              DIR_UART2[0]
#define      U2THR              DIR_UART2[0]
#define      U2DLL              DIR_UART2[0]
#define      U2IER              DIR_UART2[1]
#define      U2DLM              DIR_UART2[1]
#define      U2IIR              DIR_UART2[2]
#define      U2LCR              DIR_UART2[3]

#define      DIR_UART3    ((volatile uint32_t *) 0x4009C000UL)
#define      U3RBR              DIR_UART3[0]
#define      U3THR              DIR_UART3[0]
#define      U3DLL              DIR_UART3[0]
#define      U3IER              DIR_UART3[1]
```

```
#define      U3DLM              DIR_UART3[1]
#define      U3IIR              DIR_UART3[2]
#define      U3LCR              DIR_UART3[3]
/*****************************************************************************
**/


/*********************************DATA
TYPES*********************************/
typedef struct uart_config_tag
{
      uint8_t uart_number;
      uint32_t baudrate;
}uart_config_t;
/*****************************************************************************
**/


/*********************************PROTOTYPES*********************************
**/
void uart_init(uart_config_t * uart_cfg);
void uart_send_data(uint8_t uart, uint8_t * data, uint16_t data_size);
/*****************************************************************************
**/


/*********************************VARIABLES**********************************
**/
uint8_t tx_in_progress[3];
extern xQueueHandle queue_uart_0_tx;
extern xQueueHandle queue_uart_0_rx;
extern xQueueHandle queue_uart_1_tx;
extern xQueueHandle queue_uart_1_rx;
extern xQueueHandle queue_uart_2_tx;
extern xQueueHandle queue_uart_2_rx;
/*****************************************************************************
**/

void uart_init(uart_config_t * uart_cfg)
{
      uint32_t Fdiv = (SystemCoreClock / 16) / uart_cfg->baudrate;

      switch(uart_cfg->uart_number)
      {
            case 0:
                  PCONP |= 0x01<<3;
                  PCLKSEL0 &= ~(0x03<<6);
                  PCLKSEL0 |= (0x01<<6);
                  set_pinsel(0, 2, 1);
                  set_pinsel(0, 3, 1);
                  U0LCR = 0x83;
                  U0DLM = Fdiv/256;
                  U0DLL = Fdiv%256;
                  U0LCR = 0x03;
                  U0IER = 0x03;
                  ISER0 |= (1<<5);
                  break;

            case 1:
```

```
                PCONP |= 0x01<<4;
                PCLKSEL0 &= ~(0x03<<8);
                PCLKSEL0 |= (0x01<<8);
                set_pinsel(0, 15, 1);
                set_pinsel(0, 16, 1);
                U1LCR = 0x83;
                U1DLM = Fdiv/256;
                U1DLL = Fdiv%256;
                U1LCR = 0x03;
                U1IER = 0x03;
                ISER0 |= (1<<6);
                break;

        case 2:
                PCONP |= 0x01<<24;
                PCLKSEL1 &= ~(0x03<<16);
                PCLKSEL1 |= (0x01<<16);
                set_pinsel(0, 10, 1);
                set_pinsel(0, 11, 1);
                U2LCR = 0x83;
                U2DLM = Fdiv/256;
                U2DLL = Fdiv%256;
                U2LCR = 0x03;
                U2IER = 0x03;
                ISER0 |= (1<<7);
                break;
    }
}


void uart_send_data(uint8_t uart, uint8_t * data, uint16_t data_size)
{
    uint16_t data_index=0;
    uint8_t byte;

    switch(uart)
    {
        case 0:
            while(data_index < data_size)
            {
                xQueueSend(queue_uart_0_tx, &(data[data_index]),
portMAX_DELAY);
                data_index++;
            }

            if (tx_in_progress[0] == 0)
            {
                tx_in_progress[0]=1;
                xQueueReceive(queue_uart_0_tx, &byte,
portMAX_DELAY);
                U0THR = byte;
            }
            break;

        case 1:
            while(data_index < data_size)
            {
                xQueueSend(queue_uart_1_tx, &(data[data_index]),
portMAX_DELAY);
```

```
                        data_index++;
                }

                if (tx_in_progress[1] == 0)
                {
                        tx_in_progress[1]=1;
                        xQueueReceive(queue_uart_1_tx, &byte,
portMAX_DELAY);
                        U1THR = byte;
                }
                break;

        case 2:
                while(data_index < data_size)
                {
                        xQueueSend(queue_uart_2_tx, &(data[data_index]),
portMAX_DELAY);
                        data_index++;
                }

                if (tx_in_progress[2] == 0)
                {
                        tx_in_progress[2]=1;
                        xQueueReceive(queue_uart_2_tx, &byte,
portMAX_DELAY);
                        U2THR = byte;
                }
                break;
        }
}


void UART0_IRQHandler (void)
{
        uint8_t iir;
        uint8_t tx_byte;
        uint8_t rx_byte;
        int ret;
        portBASE_TYPE HigherPriorityTaskWoken = 0;

        iir = U0IIR;

        if (iir & 0x02)
        {
                ret= xQueueReceiveFromISR(queue_uart_0_tx, &tx_byte,
&HigherPriorityTaskWoken);

                if (ret == pdFALSE)
                {
                        tx_in_progress[0] = 0;
                }
                else
                {
                        U0THR = tx_byte;
                }
        }

        if (iir & 0x04)
        {
```

```
			rx_byte = U0RBR;
			xQueueSendFromISR(queue_uart_0_rx, &rx_byte,
&HigherPriorityTaskWoken);
		}

		portEND_SWITCHING_ISR(HigherPriorityTaskWoken);
}


void UART1_IRQHandler (void)
{
		uint8_t iir;
		uint8_t tx_byte;
		uint8_t rx_byte;
		int ret;
		portBASE_TYPE HigherPriorityTaskWoken = 0;

		iir = U1IIR;

		if (iir & 0x02)
		{
			ret= xQueueReceiveFromISR(queue_uart_1_tx, &tx_byte,
&HigherPriorityTaskWoken);

			if (ret == pdFALSE)
			{
				tx_in_progress[1] = 0;
			}
			else
			{
				U1THR = tx_byte;
			}
		}

		if (iir & 0x04)
		{
			rx_byte = U1RBR;
			xQueueSendFromISR(queue_uart_1_rx, &rx_byte,
&HigherPriorityTaskWoken);
		}

		portEND_SWITCHING_ISR(HigherPriorityTaskWoken);
}


void UART2_IRQHandler (void)
{
		uint8_t iir;
		uint8_t tx_byte;
		uint8_t rx_byte;
		int ret;
		portBASE_TYPE HigherPriorityTaskWoken = 0;

		iir = U2IIR;

		if (iir & 0x02)
		{
			ret= xQueueReceiveFromISR(queue_uart_2_tx, &tx_byte,
&HigherPriorityTaskWoken);
```

```
			if (ret == pdFALSE)
			{
				tx_in_progress[2] = 0;
			}
			else
			{
				U2THR = tx_byte;
			}
		}

		if (iir & 0x04)
		{
			rx_byte = U2RBR;
			xQueueSendFromISR(queue_uart_2_rx, &rx_byte,
&HigherPriorityTaskWoken);
		}

		portEND_SWITCHING_ISR(HigherPriorityTaskWoken);
}
```

UART.h

```
/*********************************INCLUDES*********************************
**/
#include <stdint.h>
/*************************************************************************
**/


/*********************************DATA
TYPES*********************************/
typedef struct uart_config_tag
{
	uint8_t uart_number;
	uint32_t baudrate;
}uart_config_t;
/*************************************************************************
**/


/*********************************PROTOTYPES*******************************
**/
void uart_init(uart_config_t * uart_cfg);
void uart_send_data(uint8_t uart, uint8_t * data, uint16_t data_size);
/*************************************************************************
**/
```

ADC.c

```
/**********************************INCLUDES**********************************
**/
#include <stdint.h>
#include "../inc/GPIO.h"
#include "../inc/PET_BOARD.h"

#include "../inc/FreeRTOSConfig.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
/*****************************************************************************
**/


/**********************************DEFINES**********************************
**/
#define PCONP                   (*((volatile uint32_t *) 0x400FC0C4UL))
#define      PCLKSEL            ((volatile uint32_t *) 0x400FC1A8UL)
#define      PCLKSEL0     PCLKSEL[0]
#define      PCLKSEL1     PCLKSEL[1]

#define      AD0CR              (*((volatile uint32_t *) 0x40034000))

#define      AD0DR              ((volatile uint32_t *) 0x40034010)
#define      AD0DR0             AD0DR[0]
#define      AD0DR1       AD0DR[1]
#define      AD0DR2       AD0DR[2]
#define      AD0DR3       AD0DR[3]
#define      AD0DR4       AD0DR[4]
#define      AD0DR5       AD0DR[5]
#define      AD0DR6       AD0DR[6]
#define      AD0DR7       AD0DR[7]

#define      AD0INTEN     (*((volatile uint32_t *) 0x4003400C))

#define      IPR                ((volatile uint32_t *)0xE000E400UL)
#define      IPR0               IPR[0]
#define      IPR1               IPR[1]
#define      IPR2               IPR[2]
#define      IPR3               IPR[3]
#define      IPR4               IPR[4]
#define      IPR5               IPR[5]
#define      IPR6               IPR[6]
#define      IPR7               IPR[7]
#define      IPR8               IPR[8]

#define      ISER               ((volatile uint32_t *)0xE000E100UL)
#define      ICER               ((volatile uint32_t *)0xE000E180UL)
#define      ISER0              ISER[0]
#define      ISER1              ISER[1]
#define      ICER0              ICER[0]
#define      ICER1              ICER[1]
/*****************************************************************************
**/
```

```
/*********************************PROTOTYPES*********************************
**/
void adc_init(void);
void adc_soc(void);
void adc_eoc(void);
/****************************************************************************
**/


/*********************************VARIABLES*********************************
**/
extern xQueueHandle queue_adc;
/****************************************************************************
**/

void adc_init(void)
{
	PCONP |= (1 << 12);
	AD0CR |= (1 << 21);

	PCLKSEL0 &= ~(0x03 << 24);
	PCLKSEL0 |= (0x03 << 24);

	set_pinsel(ADC_PORT, ADC_PIN, 3);
	set_pinmode(ADC_PORT, ADC_PIN, 2);

	AD0CR &= ~(0xFFFF);
	AD0CR |= (1 << ADC_CHANNEL);

	AD0INTEN &= ~(0x01<<8);
	AD0INTEN |= (0x01<< ADC_CHANNEL);

	IPR5 &= (0x1F << 19);
	IPR5 |= (0x1F << 19);

	ISER0 |= (0x01 << 22);
}


void ADC_IRQHandler (void)
{
	portBASE_TYPE HigherPriorityTaskWoken = 0;
	unsigned int valor_adc[7]; // Defino un vector valor_adc para guardar
los valores de los registros AD0DRx.
	volatile uint16_t valor_final_adc;

	valor_adc[5] = AD0DR5;                              // DODR5 GUARDA EL
VALOR DE CUENTA DEL ADC PERTENECIENTE AL  POTENCIOMETRO

	valor_adc[5] = valor_adc[5] & 0x0000FFF0;
	valor_final_adc = valor_adc[5]>>4;      // Paso los valores a la
variable global valor_final_adc
	xQueueSendFromISR(queue_adc, &valor_final_adc,
&HigherPriorityTaskWoken);

	portEND_SWITCHING_ISR(HigherPriorityTaskWoken);
}
```

```
void adc_soc(void)
{
	AD0CR |= (1 << 21); // Energizamos el ADC.
	AD0CR |= (1 << 24); // Solicitamos conversion
}


void adc_eoc(void)
{
	AD0CR &= ~(1 << 21); // Apago el conversor
}
```

ADC.h

```
/*******************************PROTOTYPES*******************************
**/
void adc_init(void);
void adc_soc(void);
void adc_eoc(void);
/************************************************************************
**/
```

GPIO.c

```
/*********************************INCLUDES*********************************
**/
#include <stdint.h>
#include "../inc/GPIO.h"

#include "../inc/FreeRTOSConfig.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
/*************************************************************************
**/


/*********************************DEFINES*********************************
**/
#define      BUZZER_OFF         0
#define      LOW_BAT                        1
/*************************************************************************
**/


/*********************************DATA
TYPES*********************************/
typedef struct buzzer_config_tag
{
      uint8_t port;
      uint8_t pin;
}buzzer_config_t;
/*************************************************************************
**/


/*********************************PROTOTYPES*********************************
**/
void buzzer_init(buzzer_config_t * buzzer_cfg);
void buzzer_set(uint8_t status);
/*************************************************************************
**/


/*********************************VARIABLES*********************************
**/
static uint8_t buzzer_port;
static uint8_t buzzer_pin;

extern xQueueHandle queue_buzzer;
/*************************************************************************
**/

void buzzer_init(buzzer_config_t * buzzer_cfg)
{
      buzzer_port = buzzer_cfg->port;
      buzzer_pin = buzzer_cfg->pin;

      set_pinsel(buzzer_port, buzzer_pin, 0);
```

```
	set_dir(buzzer_port, buzzer_pin, 1);
}


void buzzer_set(uint8_t status)
{
	xQueueSend(queue_buzzer, &status, 0);
}


void task_buzzer(void * a)
{
	static uint8_t status = BUZZER_OFF;

	while (1)
	{
		xQueueReceive(queue_buzzer, &status, portMAX_DELAY);

		if (status == BUZZER_OFF)
		{
			set_pin(buzzer_port, buzzer_pin, 1);
		}
		else
		{
			set_pin(buzzer_port, buzzer_pin, 0);
			vTaskDelay(150 / portTICK_RATE_MS);
			set_pin(buzzer_port, buzzer_pin, 1);
			vTaskDelay(150 / portTICK_RATE_MS);
			set_pin(buzzer_port, buzzer_pin, 0);
			vTaskDelay(150 / portTICK_RATE_MS);
			set_pin(buzzer_port, buzzer_pin, 1);
		}
	}
}
```

GPIO.h

```
/*********************************INCLUDES*********************************
**/
#include <stdint.h>
/*************************************************************************
**/

/*********************************PROTOTYPES*******************************
**/
void set_pinsel(uint8_t port, uint8_t pin, uint8_t function);
void set_pinmode(uint8_t port, uint8_t pin, uint8_t mode);
void set_mode_od( uint8_t port , uint8_t pin , uint8_t dir);
void set_dir(uint8_t port, uint8_t pin, uint8_t dir);
void set_pin(uint8_t port, uint8_t pin, uint8_t state);
uint8_t get_pin(uint8_t port, uint8_t pin);
void toggle_pin (uint8_t port, uint8_t pin);
/*************************************************************************
**/
```

BUZZER.c

```
/*********************************INCLUDES*********************************
**/
#include <stdint.h>
#include "../inc/GPIO.h"

#include "../inc/FreeRTOSConfig.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
/*************************************************************************
**/


/*********************************DEFINES*********************************
**/
#define      BUZZER_OFF         0
#define      LOW_BAT                        1
/*************************************************************************
**/


/*********************************DATA
TYPES*********************************/
typedef struct buzzer_config_tag
{
      uint8_t port;
      uint8_t pin;
}buzzer_config_t;
/*************************************************************************
**/


/*********************************PROTOTYPES*********************************
**/
void buzzer_init(buzzer_config_t * buzzer_cfg);
void buzzer_set(uint8_t status);
/*************************************************************************
**/


/*********************************VARIABLES*********************************
**/
static uint8_t buzzer_port;
static uint8_t buzzer_pin;

extern xQueueHandle queue_buzzer;
/*************************************************************************
**/

void buzzer_init(buzzer_config_t * buzzer_cfg)
{
      buzzer_port = buzzer_cfg->port;
      buzzer_pin = buzzer_cfg->pin;

      set_pinsel(buzzer_port, buzzer_pin, 0);
```

```
	set_dir(buzzer_port, buzzer_pin, 1);
}


void buzzer_set(uint8_t status)
{
	xQueueSend(queue_buzzer, &status, 0);
}


void task_buzzer(void * a)
{
	static uint8_t status = BUZZER_OFF;

	while (1)
	{
		xQueueReceive(queue_buzzer, &status, portMAX_DELAY);

		if (status == BUZZER_OFF)
		{
			set_pin(buzzer_port, buzzer_pin, 1);
		}
		else
		{
			set_pin(buzzer_port, buzzer_pin, 0);
			vTaskDelay(150 / portTICK_RATE_MS);
			set_pin(buzzer_port, buzzer_pin, 1);
			vTaskDelay(150 / portTICK_RATE_MS);
			set_pin(buzzer_port, buzzer_pin, 0);
			vTaskDelay(150 / portTICK_RATE_MS);
			set_pin(buzzer_port, buzzer_pin, 1);
		}
	}
}
```

BUZZER.h

```
/*********************************INCLUDES*********************************
**/
#include <stdint.h>
/*************************************************************************
**/


/*********************************DEFINES**********************************
**/
#define      BUZZER_OFF          0
#define      LOW_BAT                          1
/*************************************************************************
**/


/*********************************DATA
TYPES*********************************/
typedef struct buzzer_config_tag
{
      uint8_t port;
      uint8_t pin;
}buzzer_config_t;
/*************************************************************************
**/


/*********************************PROTOTYPES*******************************
**/
void buzzer_init(buzzer_config_t * buzzer_cfg);
void buzzer_set(uint8_t status);
/*************************************************************************
**/
```

RGB.c

```
/*********************************INCLUDES*********************************
**/
#include <stdint.h>
#include "../inc/GPIO.h"

#include "../inc/FreeRTOSConfig.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
/*************************************************************************
**/


/*********************************DEFINES*********************************
**/
#define OFF               0
#define BLUE        1 // NO GPS
#define      GREEN         2 // OK
#define      CYAN          3
#define      RED                 4 // NO GSM AND NO GPS
#define      MAGENTA       5 // NO SAFE ZONE DATA
#define YELLOW      6 // NO GSM
#define WHITE             7
/*************************************************************************
**/


/*********************************DATA
TYPES*********************************/
typedef struct rgb_config_tag
{
      uint8_t red_port;
      uint8_t red_pin;
      uint8_t green_port;
      uint8_t green_pin;
      uint8_t blue_port;
      uint8_t blue_pin;
}rgb_config_t;
/*************************************************************************
**/


/*********************************PROTOTYPES*********************************
**/
void rgb_init(rgb_config_t * rgb_cfg);
void rgb_set(uint8_t colour);
/*************************************************************************
**/


/*********************************VARIABLES*********************************
**/
static uint8_t red_port;
static uint8_t red_pin;
static uint8_t green_port;
```

```
static uint8_t green_pin;
static uint8_t blue_port;
static uint8_t blue_pin;

extern xQueueHandle queue_rgb;
/*****************************************************************************
**/

void rgb_init(rgb_config_t * rgb_cfg)
{
	red_port = rgb_cfg->red_port;
	red_pin = rgb_cfg->red_pin;
	green_port = rgb_cfg->green_port;
	green_pin = rgb_cfg->green_pin;
	blue_port = rgb_cfg->blue_port;
	blue_pin = rgb_cfg->blue_pin;

	set_pinsel(red_port, red_pin, 0);
	set_dir(red_port, red_pin, 1);

	set_pinsel(green_port, green_pin, 0);
	set_dir(green_port, green_pin, 1);

	set_pinsel(blue_port, blue_pin, 0);
	set_dir(blue_port, blue_pin, 1);
}


void rgb_set(uint8_t colour)
{
	xQueueSend(queue_rgb, &colour, 0);
}


void task_rgb(void * a)
{
	static uint8_t colour;

	while (1)
	{
		xQueueReceive(queue_rgb, &colour, portMAX_DELAY);

		switch(colour)
		{
			case OFF:
				set_pin(red_port, red_pin, 0);
				set_pin(green_port, green_pin, 0);
				set_pin(blue_port, blue_pin, 0);
				break;

			case BLUE:
				set_pin(red_port, red_pin, 0);
				set_pin(green_port, green_pin, 0);
				set_pin(blue_port, blue_pin, 1);
				break;

			case GREEN:
				set_pin(red_port, red_pin, 0);
				set_pin(green_port, green_pin, 1);
```

```
                        set_pin(blue_port, blue_pin, 0);
                        break;

                case CYAN:
                        set_pin(red_port, red_pin, 0);
                        set_pin(green_port, green_pin, 1);
                        set_pin(blue_port, blue_pin, 1);
                        break;

                case RED:
                        set_pin(red_port, red_pin, 1);
                        set_pin(green_port, green_pin, 0);
                        set_pin(blue_port, blue_pin, 0);
                        break;

                case MAGENTA:
                        set_pin(red_port, red_pin, 1);
                        set_pin(green_port, green_pin, 0);
                        set_pin(blue_port, blue_pin, 1);
                        break;

                case YELLOW:
                        set_pin(red_port, red_pin, 1);
                        set_pin(green_port, green_pin, 1);
                        set_pin(blue_port, blue_pin, 0);
                        break;

                case WHITE:
                        set_pin(red_port, red_pin, 1);
                        set_pin(green_port, green_pin, 1);
                        set_pin(blue_port, blue_pin, 1);
                        break;
        }
    }
}
```

RGB.h

```
/*********************************INCLUDES*********************************
**/
#include <stdint.h>
/*************************************************************************
**/


/*********************************DEFINES**********************************
**/
#define OFF               0
#define BLUE        1
#define      GREEN         2
#define      CYAN          3
#define      RED                 4
#define      MAGENTA       5
#define YELLOW      6
#define WHITE             7
/*************************************************************************
**/


/*********************************DATA
TYPES*********************************/
typedef struct rgb_config_tag
{
       uint8_t red_port;
       uint8_t red_pin;
       uint8_t green_port;
       uint8_t green_pin;
       uint8_t blue_port;
       uint8_t blue_pin;
}rgb_config_t;
/*************************************************************************
**/


/*********************************PROTOTYPES*******************************
**/
void rgb_init(rgb_config_t * rgb_cfg);
void rgb_set(uint8_t colour);
/*************************************************************************
**/
```

PET_BOARD.h

```
/*********************************DEFINES*************************************
**/
#define      BASEBOARD    0
#define      TDII               1

#define      PET_BOARD    TDII

#if PET_BOARD == BASEBOARD
      #define      RGB_RED_PORT       2
      #define RGB_RED_PIN             3
      #define      RGB_GREEN_PORT     2
      #define RGB_GREEN_PIN           2
      #define      RGB_BLUE_PORT             2
      #define      RGB_BLUE_PIN       1
      #define BUZZER_PORT             0
      #define BUZZER_PIN              28
      #define      UART_DEBUG                0
      #define      UART_COMM                       1
      #define ADC_PORT                        1
      #define      ADC_PIN                                 31
      #define      ADC_CHANNEL               5
#elif PET_BOARD == TDII
      #define      RGB_RED_PORT       0
      #define RGB_RED_PIN             2
      #define      RGB_GREEN_PORT     0
      #define RGB_GREEN_PIN           22
      #define      RGB_BLUE_PORT             0
      #define      RGB_BLUE_PIN       3
      #define BUZZER_PORT             2
      #define      BUZZER_PIN                0
      #define      UART_DEBUG                2
      #define      UART_COMM                       1
      #define ADC_PORT                        1
      #define      ADC_PIN                                 31
      #define      ADC_CHANNEL               5
#endif
/*****************************************************************************
**/
```