UNIVERSITAT POLITÈCNICA DE VALÈNCIA

CAMPUS DE GANDIA

GRADO EN TECNOLOGIAS INTERACTIVAS

Microprocesadores y Acondicionadores de señal **Examen del 17 de enero de 2025**

Apellidos: Aula: Columna: Fila:

Nombre: DNI:

Conteste a los diferentes apartados en las casillas habilitadas para la respuesta. Puede entregar hojas adicionales si lo considera necesario. Se proporciona, junto con el examen, unos anexos de funciones que pueden resultarle útiles para completar los apartados que requieran codificar.

EJERCICIO 1 (1 Puntos)

Se desea capturar la señal analógica de salida de un sensor que nos proporciona un **valor máximo de 3.3 V**. Se utiliza el **ADC1 (conectado al GPIO35)** del ESP32 con una **precisión de 11 bits**.

Apartado 1.1) (0.5 puntos) Configura el ADC1 utilizando la librería de **esp-idf** proporcionadas en el **Anexo 1**. Incluye la variable adc_valor con la lectura del canal específico del ADC1.

Apartado 1.2) (0.5 puntos) Calcula el valor digital que leerá el ADC1 cuando se tenga una señal de salida del sensor de 1.8 V a la entrada del mismo.

EJERCICIO 2 (2 Puntos)

Se desea **configurar** el **Timer 3** del ESP32 utilizando las funciones del entorno **esp-idf** para que genere una **interrupción periódica cada 15 ms**. Para ello se realizarán los siguientes pasos:

Apartado 2.1) (0.5 puntos) Partiendo del reloj de frecuencia 80MHz (**fclk**), y un valor de divisor de frecuencia (**P**) de 80, **hallar el valor final de la cuenta (V_{alarm})** que causará la interrupción. Detallar los cálculos realizados.

Apartado 2.2) (1.5 puntos) Codificar la función de configuración del timer siguiendo los pasos indicados en el programa dado. En el **Anexo 2** se facilitan las funciones necesarias y la estructura de configuración **timer_config_t**.

```
static void fcn_timer3_init()
{
/*Declarar estructura conf de tipo timer_config_t y sus datos públicos conf.xxx*/




/*Inicializar y Configurar el timer3*/

/*Parar el Contador del Timer*/

/*Cargar el valor inicial de la cuenta */

/*Fijar el valor de la alarma*/

/*Habilitar la interrupción del Timer*/

/*Registrar el manejador de la interrupción ISR_Timer3*/



/*Iniciar el Contador del timer*/

}
```

EJERCICIO 3 (1.5 Puntos)

Se va a configurar la **UART1 del ESP32** para comunicarnos con un dispositivo GPS a una velocidad de transmisión de 9600 baudios, para transmitir 1 byte, con bit de paridad par ($P_b = 0$) y con dos bits de stop.

Apartado 3.1) (0.5 puntos) Configurar el sistema utilizando la **librería Serial de Arduino** proporcionada en el **Anexo 3**.

Apartado 3.2) (0.5 puntos) Dibuja la trama configurada en el apartado anterior explicando claramente qué es cada uno de los bits que se están transmitiendo.

Apartado 3.3) (0.5 puntos) Calcula el Throughput (bits transmitidos por segundo)

EJERCICIO 4 (0.5 Puntos)

La siguiente figura muestra el esquema de lectura desde un dispositivo esclavo utilizando el **bus I2C**. Indicar claramente:

- Qué es cada uno de los bits o bytes que se están transmitiendo.
- Quién (maestro o esclavo) se encarga de poner cada uno de esos bits.

Lectura del esclavo

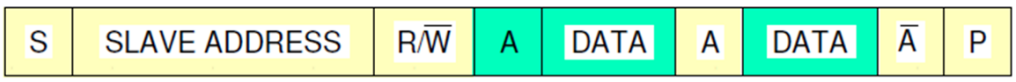

Figura 1. Trama que se envía para leer datos de un dispositivo esclavo

EJERCICIO 5 (1.5 Puntos)

Se desea **leer 2 bytes** almacenados en el registro 0x66 de un dispositivo conectado a través del bus I2C, cuya dirección es 0xA3.

Apartado 5.1.) (0.5 Puntos) Define las direcciones del registro y del dispositivo y declara la función **read_2bytes**.

```
/*Define las direcciones del registro y del dispositivo




/*Declara la función read_2bytes
uint16_t read_2bytes(                                                    );
```

Apartado 5.2) (1 Punto) Realiza la llamada a la función **read_2bytes** para leer los 2 bytes almacenados en la variable dato_leido. Tenga en cuenta que en el dispositivo primero se debe escribir la dirección del registro que se desea leer y después leer el registro. Utiliza las funciones de la **librería Wire.h de Arduino** que se proporcionan en el **Anexo 4**.

```
uint16_t read_2bytes(                                                    ) {
  uint16_t dato_leido;









}
```

EJERCICIO 6 (3.5 Puntos)

Se desea implementar un sistema de control de humedad y temperatura basado en el sensor DHT11 y el SOC ESP32.

El esquema de la arquitectura software consta de una rutina de atención a interrupciones (ISR_EXT) y 2 tareas del FreeRTOS.

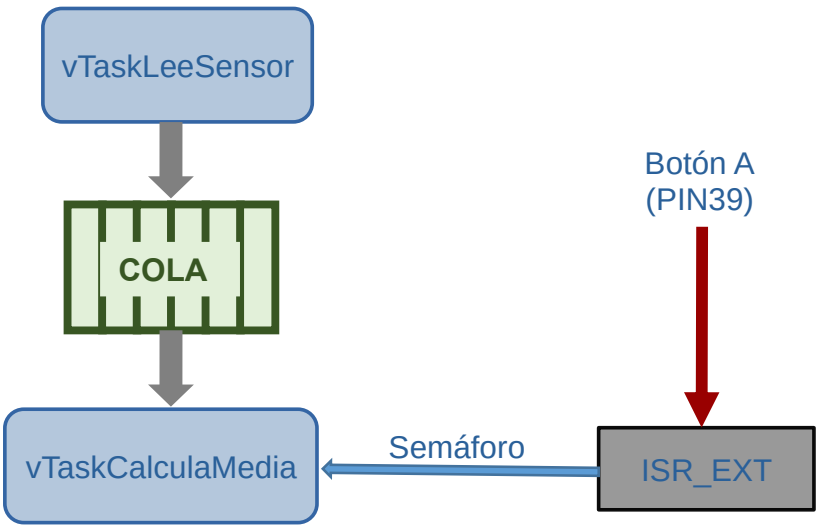

Figura 2. Sistema utilizado formado por una rutina de atención a la interrupción y dos tareas FreeRTOS.

El comportamiento del sistema es el siguiente:

- Tarea **vTaskLeeSensor**: Lee los valores de temperatura y humedad cada 10 minutos utilizando la función **"f_obtener_valores_del_sensor"** y el dato obtenido se devuelve en una estructura del tipo *type_data_sensor*. Esta estructura se almacena en una cola, la cual es consumida por la tarea **vTaskCalculaMedia** cuando se produce una interrupción externa.
    - Función utilizada: *type_data_sensor* **f_obtener_valores_del_sensor**()
    - Declaración de la estructura *type_data_sensor*:

```
typedef struct{
        uint8_t id_sender;      // Identificador del sensor
        float data_hum;         // Dato leído humedad
        float data_temp;        // Dato leído temperatura
}type_data_sensor;
```

- Rutina **ISR_EXT**: activa (da) un semáforo binario cuando un operario de planta presiona el botón A del ESP32.

- Tarea **vTaskCalculaMedia**: Cuando recibe el semáforo binario, calcula el valor medio de humedad y temperatura de las lecturas almacenadas en la cola. Para ello, utiliza la función "**f_calcula_media**", la cual devuelve el resultado en la estructura del tipo *type_output_result*
    - Función utilizada: *type_output_result* **f_calcula_media**(type_data_sensor data, UbaseType_t queue_length)
    - Declaración de la estructura *type_output_result*:

```
typedef struct{
        float avg_hum;          // Media de mediciones de humedad en cola
        float avg_temp;         // Media de mediciones de temperatura en cola
}type_output_result;
```

Se pide que escriba el código correspondiente a las siguientes partes del programa para que se implemente el comportamiento descrito anteriormente. Suponga que la estructura y las funciones mencionadas anteriormente ya están declaradas. Para cada parte debe indicar dónde debe escribirse (en el setup, loop o fuera de ambos) las líneas de código que se requieran. Utiliza las funciones de la **librería de Arduino y de FreeRTOS** que se proporcionan en los **Anexos 5 y 6**.

Apartado 6.1) (0.3 Puntos) Creación de las 2 tareas con un tamaño de la pila de 4096 bytes y con prioridad 4.

Apartado 6.2) (0.3 Puntos) Creación del semáforo binario y su manejador.

Apartado 6.3) (0.3 Puntos) Creación de la cola de tamaño TAM_MAX_COLA = 100 y su manejador.

Apartado 6.4) (0.5 Puntos) Completa las partes que se indican de la tarea **vTaskLeeSensor**.

```
void vTaskLeeSensor(void *pvParameters) {
        type_data_sensor datos;
        while(1) {
                /*Leer los datos de temperatura y humedad del sensor
                 datos =

                printf("vTaskLeeSensor: %.2f\n", datos.data_hum);
                printf("vTaskLeeSensor: %.2f\n", datos.data_temp);
                printf("\n");
```

```
                /*Enviar datos a la cola




                /* Bloquear tarea durante 10 min



}
```

Apartado 6.5) (1 Punto) Completa las partes que se indican de la tarea **vTaskCalculaMedia**.

```
void vTaskCalculaMedia(void *pvParameters) {
        type_data_sensor datos[TAM_MAX_COLA];
        UBaseType_t elementos;
        type_output_result resultado;

        while(1) {
                /* Espera a recibir el semáforo binario



                        /* Obtener el número de elementos en la cola




                        for (UBaseType_t i = 0; i < elementos; i++) {
                        /*Leer datos de la cola



                        }

                        /*Calcular la media
                        resultado =


                        // Imprimir los resultados en la consola serie
                        printf("Media de Humedad: %.2f\n", resultado.avg_hum);
                        printf("Media de Temperatura: %.2f\n", resultado.avg_temp);
                }

        }

}
```

Apartado 6.6) (0.4 Puntos) Configuración del pin 39 del ESP32 para se produzca la interrupción por flanco de bajada.

Apartado 6.7) (0.4 Puntos) Crea la rutina de atención a la interrupción **ISR_EXT**.

Apartado 6.8) (0.3 Puntos) ¿Qué pasará si el operario se duerme y no presiona el botón A durante toda la noche?