Contenido

[1. Arquitectura ESP32 2](#_Toc85116074)

[1.1. Configuración del ADC 2](#_Toc85116075)

[1.2. Configuración del ADC con Timer. 3](#_Toc85116076)

[1.3. Configuración del PWM. 4](#_Toc85116077)

[1.4. Configuración de comandos por puerto serie. 5](#_Toc85116078)

[1.5. Configuración del sensor inercial por I2C. 6](#_Toc85116079)

[2. Diseño basado en tiempo real (RTOS) 8](#_Toc85116080)

# Arquitectura ESP32

La primera práctica de este cuaderno tiene como objetivos familiarizarse con el desarrollo con el microcontrolador ESP32, manejando los periféricos presentes en cualquier microcontrolador (ADC, PWM, Timer…).

Para ello tendremos que configurar los diferentes módulos de hardware en 5 puntos diferentes.

## Configuración del ADC

Lo primero que configuraremos es la lectura del ADC integrado en el kit de desarrollo del ESP32, en nuestro caso utilizando el pin 34. Para ello, tendremos que utilizar la salida de 3V3 de la placa junto a un potenciómetro, realizando las condiciones según este esquema:

Diagrama

Descripción generada automáticamente

Una vez conectado el potenciómetro, prepararemos un programa que lea el valor del ADC y lo muestre por consola cada segundo, utilizando en este caso un delay. Para ello primero creamos una función para leer el potenciómetro:

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

Desde el loop, llamaremos a esta función e imprimiremos el resultado cada segundo:

Texto

Descripción generada automáticamente

El resultado es que cada segundo se imprime por pantalla el valor del ADC, el cual varía en función de cómo se encuentre el potenciómetro.

(CAPTURAS DE LA CONSOLA)

## Configuración del ADC con Timer.

La interrupción realizada en el apartado anterior se consigue a través de un delay. Esto supone que el kit de desarrollo no realiza ningún otro tipo de función durante el tiempo establecido. Para evitar que estas interrupciones afecten al resto del programa, se utilizan los timer, que permiten temporizar las diferentes funciones sin afectar a la ejecución del resto del programa.

Mantendremos la función leePot() del apartado anterior y simplemente cambiaremos la forma de llamarla. Para ello tenemos que configurar primero las variables necesarias para utilizar el timer:

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

En este caso tenemos dos int: interruptCounter, que es un contador que se utiliza para entrar en cada interrupción del timer según cómo se configure, en nuestro caso cada segundo, y totalInterruptCounter, que será el que aumente cada segundo, reseteándose cada vez que llegue a los segundos establecidos, en nuestro caso 10 segundos.

Además, tenemos la configuración del timer (hw\_timer\_t \* timer = NULL;) y de un multiplexor que nos permite sumar a los contadores con cada interrupción del timer.

Observamos el resto de configuración necesaria para utilizar este periférico:

Texto, Carta

Descripción generada automáticamente

Se crea una función onTimer() que aumente el valor del contador en cada interrupción.

Por otra parte, en el setup realizamos el timerBegin, teniendo como argumentos en primer lugar el timer a utilizar (del 0 al 3 que son los 4 disponibles para este hardware), luego el multiplicador, en este caso 80 porque la frecuencia de trabajo del ESP32 es de 80MHz y de esta forma podemos trabajar en microsegundos, y, por último, cómo cuenta este temporizador, siendo true si cuenta hacia arriba o false si cuenta hacia abajo. (del 1 al 10 o del 10 al 1)

Luego le añadimos la interrupción a la función creada con el timerAttachInterrupt, teniendo como argumentos el timer creado, la función onTimer creada y el tipo de interrupción, de flanco (true) o de nivel (false).

Lo siguiente es el timerAlarmWrite, que establece el número de ticks del procesador (1MHz son 1 millón de ticks por segundo) que se utiliza para las interrupciones, por ello utilizamos el valor de 1000000, y el true establece que el contador se reinicia y, por tanto, es periódico.

Por último, habilitamos con el timerAlarmEnable la interrupción creada.

Una vez configurado el timer, lo utilizamos en el loop para llamar a la función leePot() cada 10 segundos:

Texto, Carta

Descripción generada automáticamente

De esta forma, en cada interrupción, configurada como se ha explicado a 1 segundo, se resetea el valor de interrupciones para volver a parar cada segundo y se aumenta el valor de interrupciones total. Cuando el valor de interrupciones total llega a 10, se lee e imprime el valor del potenciómetro y se reinicia el valor de interrupciones total.

El resultado es que cada 10 segundos se imprime por pantalla el valor del ADC, que sigue dependiendo del potenciómetro.

(CAPTURAS CONSOLA)

## Configuración del PWM.

En este apartado aprenderemos a configurar las salidas PWM del ESP32. Tendremos que configurar un PWM con una frecuencia de 5kHz y un ciclo de trabajo proporcional a la lectura del ADC.

El pin seleccionado en nuestro caso será el 12, configurándolo de la siguiente forma:

Texto

Descripción generada automáticamente

De esta forma, el canal del PWM será el 0, la resolución será de 12 bits, la misma que el ADC, y la frecuencia serán 5000 Hz, 5kHz. El LED\_GPIO será el pin de salida del PWM.

En el setup, añadimos la configuración de forma que el canal PWM se saque por el pin establecido y que el PWM se configure con los valores mencionados anteriormente:



Para establecer el ciclo de trabajo en función del ADC se utiliza la siguiente función:



Con esta función, le establecemos al canal del PWM el ciclo de trabajo leído en el ADC.

El resultado lo podemos ver midiendo con un osciloscopio las salidas 12 y 34, el del PWM y el del ADC respectivamente.

(CAPTURAS OSCILOSCOPIO)

## Configuración de comandos por puerto serie.

En este punto tendremos que configurar el programa de forma que, según la petición realizada por pantalla, nuestro controlador realice diferentes acciones, siendo estas:

* 1. ADC: Envíe la lectura del ADC actual
  2. ADC(x): envíe la lectura del ADC cada x segundos. Si x=0, deja de mandar datos
  3. PWM(x): comanda el duty cycle del módulo PWM con números del 0 al 9.

Para ello tenemos que utilizar las diferentes funciones de los String y del Serial.

En el loop tendremos que poner un if (Serial.available() > 0 para poder introducir texto en el puerto serie, y dentro de este, las diferentes lecturas y comparaciones de dichas lecturas para acceder a las diferentes funciones:

Texto

Descripción generada automáticamente con confianza media

En este programa entraremos en diferentes secciones en función del texto que recibe el puerto serie. Además, tenemos la función funcionEscritura() que se ejecuta al principio del loop en función de los valores recibidos. En esta función podemos ejecutar las acciones requeridas para cada caso:

Texto

Descripción generada automáticamente

Se llama a las funciones ADCx y PWMx teniendo el número x recibido por el puerto serie como input en la función.

Las funciones serían las siguientes:

Texto

Descripción generada automáticamente

La función ADCx es igual a lo realizado en la sección 1.2, con la diferencia de que los segundos necesarios para la totalInterruptCounter vienen dados por el número recibido del puerto serie.

La función PWMx simplemente hace un mapeo para que los valores del 0 al 9 se transformen según la resolución (12 bits) del PWM y se escriban como ciclo de trabajo en el PWM.

El resultado es el siguiente:

(CAPTURAS CONSOLA)

## Configuración del sensor inercial por I2C.

En esta sección tendremos que configurar un sensor inercial conectado por I2C de forma que muestree la aceleración cada 100 ms y mande los datos por el puerto serie cada segundo, activando un LED durante 200 ms.

Diagrama, Esquemático

Descripción generada automáticamente

Para ello utilizaremos una librería que permita controlar el acelerómetro MPU9250, configurando el sensor según dicha librería, además del Wire para poder configurar la comunicación I2C:

Texto

Descripción generada automáticamente

Para configurar el sensor y el Wire utilizamos los siguientes comandos de la librería:

Texto

Descripción generada automáticamente

En este caso sólo iniciaremos la aceleración del sensor ya que es lo único que vamos a utilizar.

Por otra parte, como tendremos que temporizar tres funciones distintas (leer, escribir en pantalla y apagar el led después de 200 ms), añadiremos dos contadores nuevos además del totalInterruptCounter, contadorSerial y contadorLed:

Texto

Descripción generada automáticamente

Para esto creamos una función que incluya todo llamada leeSensor a la que llamaremos desde el loop:

Texto

Descripción generada automáticamente

Leeremos la aceleración del sensor con sus valores x, y, z y el módulo de la aceleración, imprimiéndolos por pantalla, cuando el contadorSerial llega a 1000 ms. Para poder medir en ms, hemos tenido que cambiar la configuración del timerAlarmWrite a 1000, en lugar de 1000000.

Además, pasados 1200 ms se apagará el Led si este estaba encendido.

Los resultados son los siguientes:

(CAPTURAS CONSOLA)

# Diseño basado en sistema operativo de tiempo real (RTOS).

Está práctica se centra en aprender el uso y conceptos asociados a un sistema operativo en tiempo real (RTOS) y diseñar un firmware basado en el mismo.

Trabajamos con una librería llamada ESP32\_FreeRtos.

De esta forma, adaptaremos el sistema desarrollado en el punto 5 de la práctica anterior a través del uso de tareas.

## Implementación con RTOS de la lectura de un sensor inercial.

La tarea a implementar utilizando RTOS es la adquisición de datos de aceleración de nuestro sensor inercial, de forma que el sensor lea la aceleración cada 100 ms y cada segundo, se imprima el valor en pantalla. Al imprimir el valor en pantalla, se debe encender durante 200 ms un led.

Lo primero que debemos hacer es configurar el sistema RTOS:

Texto

Descripción generada automáticamente

Nuestra librería FreeRTOS funciona a través de tareas, de forma que se crean diferentes tareas independientes entre sí que se ejecutan de forma continua. Para temporizar estas tareas tenemos delays para cada tarea específica, de forma que el delay sólo se aplica a una tarea, sin afectar a las demás. Por ello, crearemos una tarea para cada una de las acciones que debemos realizar, muestreo del acelerador, envio de datos al puerto serie y encendido y apagado del led, que se definen de la siguiente forma:

Texto

Descripción generada automáticamente

Además de definirlas, tenemos que configurarlas en el setup:

Texto, Carta

Descripción generada automáticamente

Se puede configurar el nombre de la variable en primer lugar y un segundo nombre utilizado para que se pueda reconocer por parte de las personas que utilicen el programa. La otra parte que necesitamos conocer es la prioridad, en nuestro caso 3 para el muestreo, 2 para el envío de datos y 1 para el led.

Preparamos las funciones, añadiendo comentada una pequeña descripción de la tarea.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Utilizamos los vTaskDelay para temporizar las acciones, en este caso el Led se espera 1000 ms para encenderse y se apaga tras otros 200 ms. Los vTaskDelay se definen en ticks, por lo que dividimos el valor en ms que queremos por el portTICK\_RATE\_MS.

De forma similar, tendremos las otras dos funciones:

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

El resultado es igual que el obtenido en la práctica 1.5, sin embargo, utilizando este tipo de sistema vemos que nos podemos evitar ocupar memoria con contadores para cada función que queremos utilizar.

(CAPTURAS CONSOLA)