Práctica 1

Introducción

Un microcontrolador pude realizar tareas al igual que el procesador de una computadora, pero de manera mucho más simple y ahorrando mucha energía, por lo que se suele utilizar en el desarrollo de aplicaciones muy específicas. En este caso, usamos uno para hacer un reloj digital que muestra la hora, puede programar una alarma, y lleva un cronómetro, todo al mismo tiempo.

Sin embargo, el microcontrolador que utilizamos en la práctica no tiene la capacidad para realizar varias funciones al mismo tiempo. En lugar de eso, hace una tarea, luego otra, y otra a una muy alta velocidad, dando así la impresión de que hace todo al mismo tiempo. A esto se le llama concurrencia.

Función 1: Reloj en tiempo real

Este reloj funciona mostrando la hora actual en displays de 7 segmentos.

Para cambiar la hora, se utilizan tres botones:

Un botón que se tiene que dejar presionado para entrar en modo de configuración y poder usar los botones de horas y minutos, otro que al ser presionado modifica las horas de manera ascendente y otro para cambiar los minutos de igual manera.

Función 2: Alarma

La alarma nos permite programar una hora en la que queremos que suene un tono a través de una bocina. Los botones para programarla funcionan de manera similar a los de la hora:

Un botón se mantiene presionado para entrar en modo de configuración de la alarma, los otros dos botones se utilizan para aumentar las horas y minutos hasta coincidir con la hora a la que queramos que se active la alarma. Al soltar el botón de configuración, los displays vuelven a mostrar el reloj digital con la hora actual correcta.

Una vez configurada, otro botón se usa para activar la alarma, y cuando está activada, se enciende el led rojo del led RGB.

Cuando la hora del reloj coincide con la hora programada de la alarma, un tono suena a través de una bocina conectada a la salida del DAC. Este sonido es generado por una onda senoidal en forma de tabla de valores como en la tarea del generador de señales.

Función 3: Cronómetro

El cronómetro cuenta desde cero y muestra minutos, segundos y centésimas de segundo. Para activarlo, usamos un botón que pone los displays en cero y un botón de “inicio” para

que empiece a contar. Podemos pausar y reanudar el cronómetro y, mientras tanto, el reloj sigue funcionando, y la alarma, si está activada, sonará cuando sea la hora.

Cómo se logra que parezca que todas las funciones trabajan al mismo tiempo

A pesar de que nuestro microcontrolador no puede trabajar en paralelo, utilizamos una estructura de código que asigna tiempos pequeños a cada tarea: un instante para mostrar la hora, otro para revisar si es la hora de la alarma, otro para actualizar el cronómetro, etc. Esto ocurre tan rápido que el usuario no nota que está cambiando entre tareas, así que parece que todo funciona al mismo tiempo. Esto se llama calendarización de tareas.

Para lograr esto utilizamos interrupciones con diferentes prioridades, siendo la más alta la de los relojes internos que controlan el tiempo mostrado en los displays.

Paso 4: Watchdog Timer (WDT)

Finalmente, implementamos un atchdog Timer. Su tarea es la de asegurarse de que el microcontrolador no se quede atrapado en algún error, en este caso un bucle infinito. En caso de que el microcontrolador dejara de responder, el watchdog lo reinicia y muestra un mensaje de error en los displays

Relación entre los 4 callbacks

Utilizamos un callback para cada puerto; GPIOA, GPIOB y GPIOC de manera que una interrupción detectara cuando se presionó algún botón en estos puertos y mande a llamar el callback del respectivo puerto con las banderas del puerto como parámetro, para esto tuvimos que cambiar la declaración del callback para las GPIOs, haciéndolos apuntadores a variables **uint32_t**.

El cuarto callback lo utilizamos para el PIT, utilizando un solo canal para el reloj, stopwatch y alarma, con este canal implementamos una función que cuenta desde microsegundos a horas para el reloj, y en el cronómetro una función que cuenta desde microsegundos a minutos, la función para la alarma recorre la tabla de valores de la función seno.

Conclusión Diego: El principal reto y a la vez mi principal aprendizaje en está practica fue el manejo de callbacks, los cuales nos sirvieron para dar cierta modularidad y organización al código ayudándonos a estructurarlo mejor. También nos permitieron manejar de mejor manera las funciones llamadas por las interrupciones y a mejorar la simulación del trabajo en paralelo. El reto que tuvimos con los callbacks fue que tuvimos que declarar los apuntadores para que apuntaran a funciones que reciban uint32_t como parámetro para mandar el GPIO necesario debido a la limitación de 4 callbacks, por lo que los callbacks quedaron limitados a funciones que reciben este tipo de argumento. A parte de esto también reforcé mis conocimientos sobre la técnica de multiplexeado para los displays y el manejo de interrupciones en puertos GPIO.

Conclusión Kenneth:

Esta práctica nos permitió entender la importancia del diseño de software por capas a la hora de programar en sistemas embebidos. Implementar el reloj, alarma y cronómetro como capas separadas hizo que el código fuera más claro, fácil de mantener y de probar. Cada capa se encargaba de una funcionalidad específica lo que facilita la legibilidad y modularidad del código, haciéndolo más fácil de editar y trabajar en el.

Este enfoque por capas nos facilitó la simular las tareas en paralelo junto con los callbacks, lo que nos permitió lograr que cada función respondiera de inmediato a eventos sin interferir con las demás. Resolver los problemas que esto implicaba nos enseñó cómo estructurar el software para obtener un sistema flexible y eficiente.