IOT – Sistemas Embebidos

Proyecto Final

Integrantes del grupo:

- Nicolás Raposo
- Martín Da Rosa
- Rafael Durán

Docentes: Nicolás Calarco y Pablo Alonso

3/7/2025

Índice

Introducción

El proyecto descrito en este informe consiste en utilizar el kit ESP32-S2 Kaluga 1 para hacer un reproductor de música que posea las siguientes instrucciones de control:

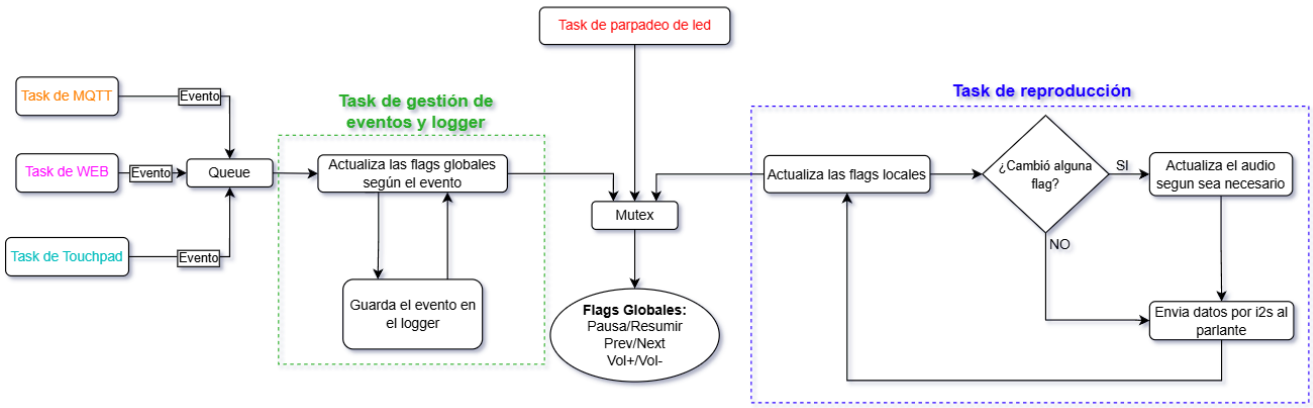
- Canción anterior y siguiente
- Subir y bajar volumen
- Pausa y resumir

Estas instrucciones del control deberán poder recibirse tanto por protocolo MQTT, a través de una página web de configuración general y mediante un touchpad. Los últimos 20 eventos enviados por estos medios serán guardados por un logger. En la web se podrán configurar los parámetros de comunicación MQTT, la configuración de conexión de la STA y la posibilidad de agregar o quitar canciones.

Diseño

Diagrama de flujo

Al igual que en otros laboratorios se hizo un diagrama de flujo para organizar las tareas que serán necesarias y ver como interactúan entre sí:



Como se puede observar, la idea general es que existen flags compartidas que afectan a la reproducción del audio, la task de reproducción se encarga de revisarlas constantemente con un mutex, actualizando el audio según sea indicado.

Estas flags globales pueden ser modificadas por las task de MQTT, WEB y el touchpad. Cuando el usuario interactúa con estas tasks, se crean eventos que son almacenados en una queue, posteriormente la task de gestión de eventos y logger guarda el tipo de evento en el logger y mediante un mutex actualiza la flag global indicada. Por último, la task de parpadeo de led se encarga de revisar solamente la flag de pausa, solo cuando el parlante esté reproduciendo música hace parpadear al led cada 250ms.

Implementación y descripción de las tasks

Task de reproducción

Para implementar la reproducción de audio se partió de unos de los ejemplos proporcionados por ESP-IDF que establece la comunicación con la placa de audio LyraT-8311A (en este caso la V1.2). El ejemplo usa el protocolo I^2C para establecer la conexión entre el chip ESP32 y el chip de la placa de audio, posteriormente, los datos digitales de audio se envían mediante el protocolo I^2S. Los archivos de audio tienen que estar en formato PCM (Pulse Code Modulation) para que la placa pueda convertir el audio digital a la señal analógica que recibirá tanto el parlante como el puerto jack.

Todo lo anterior mencionado se implementa en las funciones:

- `es8311_codec_init()`: inicializa la conexión con la placa de audio con el protocolo I^2C.
- `i2s_driver_init()`: inicializa la configuración del canal I^2S

En cuanto a las funciones requeridas, en primer lugar, la lista de canciones se implementó como un array y se navega por él mediante un índice (current_track) que se incrementa o decrementa según se reciba el evento de canción siguiente o anterior. En segundo lugar, el cambio de volumen funciona comparando el último volumen registrado (last_volume) con el volumen global (volume), si hay una diferencia, se cambia el volumen de reproducción y también se actualiza el último volumen registrado. Por último, la pausa simplemente funciona evitando que se envíen datos por el canal I^2S.

Es importante aclarar que el código de ejemplo del que se partió enviaba todo el archivo de audio de una sola vez, lo que provocaba que estos eventos no se actualizasen en vivo, por ejemplo, aunque la task de reproducción estuviese en pausa, el audio seguía saliendo por el parlante. Por eso fue necesario modificarlo para que en vez de enviar todo el audio por el canal I^2S se enviasen pequeños "paquetes" de 1KB, como referencia, las canciones pesan entre 60KB y 100KB. De esta manera se reproduce solo un poco de audio y enseguida se vuelven a revisar las flags dando la sensación de que se actualiza en vivo.

Finalmente la task `i2s_music()`revisa las flags y envía los paquetes de datos de audio.

Task de led

Esta task es sumamente simple, se encarga de revisar la flag de pausa con un mutex, cuando esta resulta ser falsa (el parlante está reproduciendo audio) hace parpadear al led de la placa en rojo, cuando resulte ser verdadera (el parlante está pausado) el led se mantiene apagado.

Task de touchpad

Para esta task se recicló parte de lo hecho en el Laboratorio 2. Mediante la función `touch_buttons_get_pressed()` se detectan cambios en el valor crudo de cada botón y si se supera un umbral configurado (TOUCH_THRESHOLD), interpreta que el botón fue presionado. Por otro lado la task `touch_event_task()` llama a la función anteriormente mencionada cada pocos milisegundos y si resulta que se detecta un evento se envía a la queue.

Task de gestión de eventos y logger

Gestión de eventos

Todos los comandos del reproductor son emitidos a través de una queue, esto permite poder agregar nuevos clientes (MQTT, Webserver y Touchpad) de manera independiente a la lógica del reproductor. Manteniendo un contrato de cómo es la estructura de los eventos que representan los comandos en el reproductor, la task de reproducción se encarga de leer dichos eventos y ejecutar los comandos.

Logger y configuración.

Tanto el logger como la configuración se guardan en dos archivos tipo JSON: config.json y logger.json. Para ello se creó una librería llamada mi_fs la cual sirve de abstracción entre el filesystem y los objetos ya cargados en variables (struct tipo logger y config). La librería mi_fs en sí únicamente sirve para leer ambos archivos y también guardarlos. Por otro lado, se creó una librería llamada mi_config, capaz de guardar callbacks a eventos de configuración. Esto es para poder reaccionar a los cambios de configuración (por ejemplo, al cambiar la url del broker mqtt, es necesario reconectarse a la nueva URL).

En resumen, mientras mi_fs únicamente se encarga del guardado y el recuperado de la información, mi_config se encarga de la notificación de los cambios de configuración.

Como utilidad para escribir y leer de memoria no volátil se utilizó littlefs, un filesystem diseñado exclusivamente para microcontroladores.
Para el correcto funcionamiento de littlefs fue necesario indicar las particiones de manera manual (mediante un partitions.csv) en el cual se encuentra la nueva partición que será utilizada por littlefs. Para los eventos del log se utilizó un array de strings de 20 espacios y un puntero de posición que itera entre 0 y 20. Esto es para hacer el comportamiento cíclico en caso de que se superen los 20 eventos, el puntero vuelve a 0 sobrescribiendo los eventos más viejos.

El evento es un string donde figura el nombre del evento y un timestamp con la hora asociada al evento. (La hora es obtenida posterior a la sincronización con el servidor NTP).

Conexion a STA y sincronizacion con NTP

Al comenzar el programa, se lee de la configuración los datos para conectarse al STA, dicha conexión es importante dado que sirva para conectarse al broker MQTT y para conectarse al pool NTP para sincronizar la hora. Se utilizo la misma librería desarrollada en laboratorios anteriores para conectarse a la red mediante wifi, solo que se le agrego un pequeño cambio, en la librería mi_config se agregó un callback que escucha el evento de cambio de configuración de los parámetros del STA para reconectar a la red en caso de que se notifique un cambio de configuración.

Task de MQTT

Primero se creó un broker de prueba en una computadora la cual se encontraba en la misma red que la placa, se utilizó para esto Mosquitto y Docker. Una vez levantado el broker se utilizó el cliente mqtt_client para conectarse desde la placa al broker. La URL que se utilizó era la IP de la computadora donde se tenía el broker levantado (la IP es la de la red que compartía con la placa).

Al conectarse se envían los eventos del logger de manera cronológica descendente y luego se queda a la escucha por nuevos eventos. Al llegar un evento se identifica a que comando corresponde y se pushea en la queue de comandos.

Los tópicos utilizados son:

- /player/control - Para escuchar nuevos comandos (bajar/subir volumen, siguiente/anterior y pausa/resumir).
- /player/logger - Para enviar los eventos registrados hasta el momento del logger inmediatamente al conectar al broker.

Al haber un cambio de configuración de la broker URL, un callback es ejecutado que vuelve a intentar conectar al broker, y una vez conectado vuelve a enviar los eventos del logger.

Task de WEB

La interfaz web del sistema permite controlar el reproductor de forma remota desde cualquier navegador conectado a la red WiFi de la placa. El servidor define una serie de endpoints, que son direcciones específicas (como /VolUp o /PlayPause) que pueden ser accedidas por la web para interactuar con el sistema. Cada endpoint responde a solicitudes HTTP de tipo POST, que se utilizan para enviar comandos como cambiar de pista, ajustar el volumen o pausar la reproducción. Cuando el servidor recibe uno de estos comandos, genera un evento y lo envía a la queue donde luego es procesado por la task de gestión de eventos ya mencionada. Además, se incluyeron endpoints GET que permiten recuperar configuraciones actuales del sistema, como la dirección del broker MQTT.

Posibles mejoras

Como se ha visto a lo largo del informe, el reproductor de música cuenta con todas las funciones principales requeridas, sin embargo, algunas ideas tuvieron que ser descartadas principalmente por la falta de tiempo y memoria. Entre ellas se encontraban:

- Añadir alguna retroalimentación visual del led frente a los eventos, por ejemplo, que cambie de color al cambiar la canción o que cambie el brillo según el volumen.
- Implementar el uso de archivos MP3 para poder utilizar canciones más largas y de mejor calidad (el formato MP3 pesa mucho menos que el PCM).
- Encriptar los mensajes con alguna codificación.
- Mejorar el uso del touchpad, haciendo la detección más precisa y permitiendo que la acción mantener presionado el botón sea válida.
- Mostrar en la web el estado actual del reproductor (estado de pausa, canción y volumen actuales, etc).

Conclusiones

En definitiva, se logró desarrollar un reproductor de música embebido con la ESP32-S2 Kaluga 1, con control vía MQTT, web y touchpad. Al igual que en laboratorios anteriores, definir claramente la estructura de conexión entre las tasks permitió trabajar de forma independiente y paralela, aprovechando las herramientas de FreeRTOS vistas en el curso. El sistema funciona correctamente, permite configuraciones dinámicas y responde adecuadamente a los comandos. Como se mencionó, todavía hay espacio para mejoras, tanto visuales como en el uso del led así como más críticas, por ejemplo la implementación de encriptado en los mensajes.