Fecha de entrega Lunes 08 de Julio de 2024.

Disclaimer This latex template has been obtained from IEEE ACCESS for educational purpose.

Identificador de notas interactivo

ETIENNE CABIAC M.[1], MATEO DE LA CUADRA C.[1]
[1]Pontificia Universidad Católica de Chile (e-mail: etienne.cabiac@uc.cl, mateodlcc@uc.cl)

SI Autorizo que mi proyecto (tal como ha sido entregado, sin nota ni comentarios de evaluación) sea publicado en un repositorio para pueda servir de guía y ser mejorado en proyectos de futuros estudiantes.

Este proyecto ha sido desarrollado bajo el curso IEE2463: Sistemas Electrónicos Programables.

ABSTRACT El proyecto presentado a continuación constituye principalmente un analizador de canciones en frecuencias. Para ello primero leemos las canciones *sampleadas* ubicadas en la tarjeta SD, luego, según la canción seleccionada utilizando el *display*, se obtiene el valor de la frecuencia más relevante y se hace sonar el *buzzer* según este resultado. Además, se implementan diferentes periféricos del *Booster Pack* que nos fue proporcionado para interactuar, permitiendo, por ejemplo, cambiar la canción seleccionada con el acelerómetro, mostrar un gráfico del micrófono para analizar nuestra voz y en general, interactuar con el software.

Para llevar a cabo el proyecto utilizamos Vivado y Vitis junto con una tarjeta Zybo Z7-10, en particular, utilizamos VHDL y C como lenguajes de programación. Además, como ya fue mencionado, usamos un *Booster Pack* que contiene múltiples periféricos con los que interactuamos. Para poder llevar a a cabo correctamente el proyecto tuvimos que manejar señales, implementar la FFT en vivado, manejar conceptos y convenciones propias de C como estructuras, arreglos, punteros, etc. así como también *floating point* (IEEE754) y PWM, por nombrar algunos.

Como resultados, fuimos capaces de llevar a cabo el proyecto en su mayoría, esto es ya que nos faltó implementar ciertas de las visiones que teníamos y lograr hacerlas de la mejor manera posible. A pesar de esto, fuimos capaces de cumplir, satisfactoriamente, con todo lo requerido.

INDEX TERMS VHDL, FFT, AXI, Vitis, Sonido, Display

I. ARQUITECTURA DE HARDWARE Y SOFTWARE (1 PUNTO)

EL proyecto se divide en dos partes principales, el *hardware* y el *software*. Comencemos por analizar el hardware, el cual se compone de 4 áreas importantes: La Zynq, las interrupciones, el análisis de frecuencia y los periféricos de la tarjeta *Booster*. Y luego pasemos a comprender el funcionamiento del Software.

A. HARDWARE

Cada una de estas áreas incluye múltiples bloques que actúan en conjunto para llegar al objetivo deseado, el cual es, interactuar con los periféricos mediante una interfaz didáctica e intuitiva, permitiéndonos escuchar una simplificación de las canciones seleccionadas.

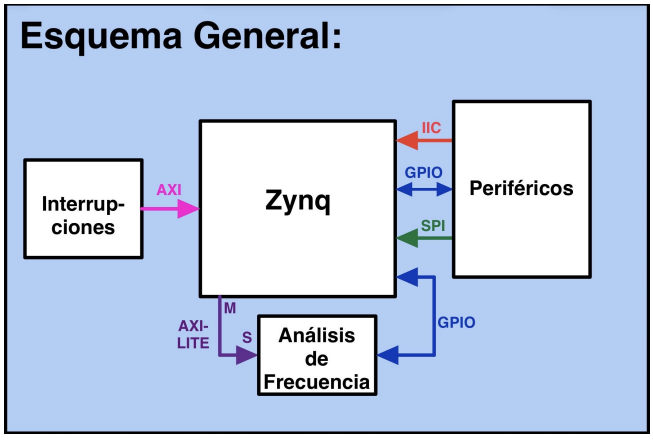

FIGURE 1: Esquema general del proyecto.

1) Zynq

Para lograr nuestro objetivo utilizamos la Zynq, que contiene un microprocesador (ARM Cortex-A9) en el cual cargamos nuestro software. En este caso, se estarían utilizando para enviar y recibir señales las cuales son procesadas y permiten

un funcionamiento lógico que cumple con nuestro objetivo. Esta se conecta con el resto de las áreas mediante diferentes tipos de conexión, como GPIO, AXI, I^2C y SPI.

2) Análisis de Frecuencia

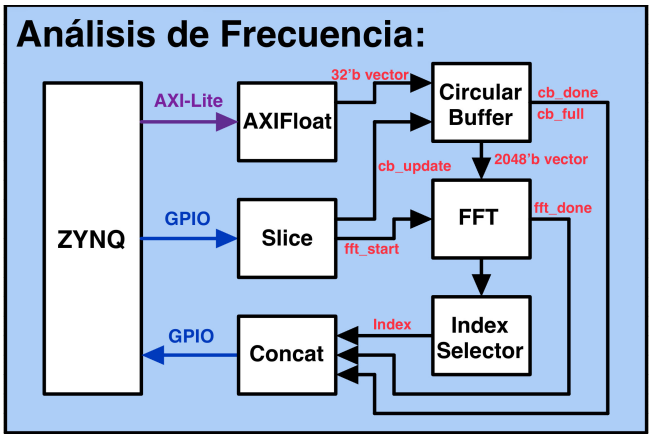

FIGURE 2: Análisis de Frecuencia.

Para poder analizar la frecuencia, lo que se hizo fue conectar la ZYNQ a nuestra lógica de hardware como mediador, en resumen, el *software* es quien se encarga (gracias a las interrupciones de los *timers*) de avisarle al '*Circular Buffer*' y a la *FFT* que pueden continuar procesando la información entregada a través del módulo AXIFloat. Finalmente, el '*Index Selector*' se encarga de informar cual es la frecuencia de mayor relevancia en ese momento, permitiendo que esta se escuche en el *Buzzer*.

3) Interrupciones

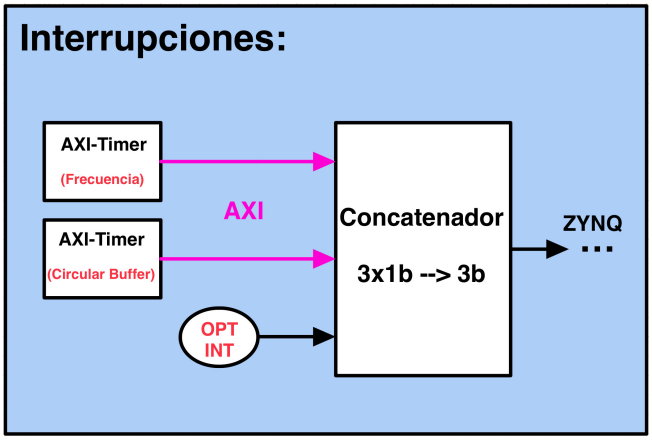

FIGURE 3: Interrupciones.

Para las interrupciones, se utilizaron 2 AXI-Timer[1], uno para cambiar la frecuencia emitida por el *buzzer* y el otro para actualizar los valores del '*Circular Buffer*' y también una interrupción proveniente del sensor de luz. Estos 3 valores son concatenados y enviados a la Zynq, en donde se analizan y se toman las acciones correspondientes.

4) Periféricos

Los periféricos utilizados son los del *Booster Pack*, en el proyecto se utilizan: el micrófono, el *buzzer*, los sensores de luz y temperatura, el *joystick*, una de las perillas, el *display* y el acelerómetro. Para conectar con el *buzzer* se utiliza un IP-Core llamado 'Buzzer', el cual se encarga de recibir información de la Zynq como la frecuencia a emitir, el valor del potenciómetro para modificar la amplitud y la variable 'MOOD' la cual se envía a través de UART y determina la escala de octavas en las cuales opera el *buzzer*. En otras palabras, se encarga de hacer sonar el *buzzer* variando su amplitud, rango de frecuencias y nota musical a emitir.

Con esto concluimos la arquitectura de Hardware y pasamos al funcionamiento del software el cual se compone por una máquina de estados principal, que llama al resto del funcionamiento.

B. SOFTWARE

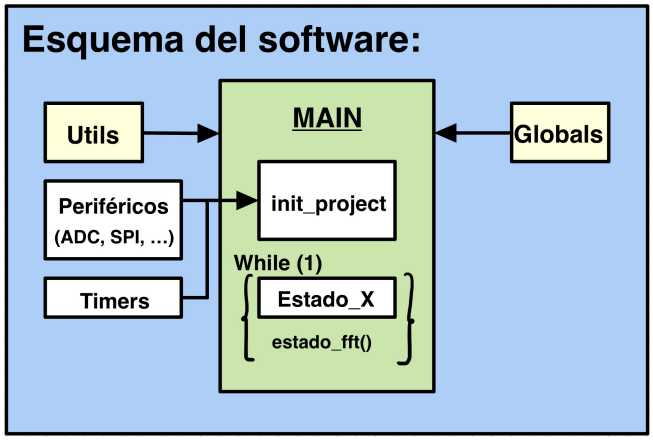

FIGURE 4: Esquema del software.

Nuestro Software se compone, de múltiples archivos, entre ellos, los más relevantes para la arquitectura de software son: (Hay que considerar que cada archivo de extensión '.c' tiene su header, de extensión '.h')

1) Globals

Estos archivos se encargan de definir todas las variables globales del proyecto así como ciertas macros y estructuras que nos fueron de utilidad a lo largo del proyecto y que, por lo general, se manejaban en múltiples archivos.

2) Utils

La diferencia con globals es que en estos archivos se creaban, principalmente, funciones que se utilizaban en múltiples ocasiones a lo largo de diferentes archivos.

[1]Fe de erratas: AXI-Timer no se comunica por AXI con el concatenador, sino con el módulo axi-peripheral

3) Main

Aquí se instancia la lógica principal, o sea, el flujo del programa. El archivo primero ejecuta la iniciación del proyecto ('init_project') y luego entra en un *loop* infinito que maneja los diferentes estados y opciones seleccionadas por el usuario. Es básicamente una máquina de estados que une los diferentes archivos implementados.

4) Init$_{Project}$

Estos archivos se encargan de instanciar el proyecto, es decir, los diferentes tipos de comunicación, las interrupciones y las configuraciones con las que se va a ejecutar el proyecto. Esto lo hace llamando a muchos archivos auxiliares como *timers*, ADC, SPI, LCD_GUI, entre muchos otros.

5) EstadoX

Existen 4 archivos (cada un con .c y .h) de este estilo, siendo uno para cada uno de los estados posibles (de 0 a 3), estos manejan la lógica propia del estado en el que nos encontramos. El estado 0 sería el menú principal donde seleccionamos a que estado queremos pasar. El estado 1 es la selección de canciones leídas desde la SD y su reproducción. Luego, el estado 2 nos muestra un gráfico de la lectura realizada por el micrófono., y por ulimo, el estado 3 no hace nada realmente interesante pero quedó allí por si teníamos tiempo para seguir desarrollando.

6) Otros

Otros archivos interesantes son: 'sdFunctions', que maneja la lógica de la SD para instanciarla, leerla, etc., 'timers' que se encarga de manejar correctamente las interrupciones para actualizar la nota emitida y aquellos utilizados para el manejo de periféricos adaptados de aquellos utilizados en el laboratorio n°10.

II. ACTIVIDADES REALIZADAS (1.5 PUNTOS)

1) Actividades Obligatorias

AO1
- *Descripción:* Utilizamos la pantalla de la tarjeta *booster* para mostrar la información que queremos. Esta se actualiza cuando lo necesitamos y logra lo requerido. Por otro lado, implementamos múltiples periféricos, por ejemplo, el micrófono, acelerómetro y sensor de luz. Para este último, modificamos los registros del periférico para habilitarlo y crear la interrupción correspondiente.
- *Nivel de Logro:* 100%. Completamente logrado, el código implementado ha compilado, se logra simular luego de la implementación, se logra generar bitstream y cargarlo en la ZYBOZ7.

AO2
- *Descripción:* Usamos 2 interrupciones derivadas de timer para poder actualizar la frecuencia que queremos emitir con el *buzzer*, en específico estas se relacionan con el *circular buffer* y la FFT, para luego repercutir en el *buzzer*. Por otro lado, utilizamos la interrupción del sensor de luz para determinar si nos encontramos en el estado 'día' o 'noche'.
- *Nivel de Logro:* 100%. Completamente logrado, el código implementado ha compilado, se logra simular luego de la implementación, se logra generar bitstream y cargarlo en la ZYBOZ7.

AO3
- *Descripción:* El IP-Core que tenemos que interactúa tanto con un periférico como con el procesador es el llamado "**Buzzer**", este depende de uno de los potenciometros para determinar la amplitud y produce el sonido en el Buzzer. El parámetro que le entregamos para su configuración es 'mood' el cual afecta la frecuencia que suena sumándole un valor dependiente de dicha variable.
- *Nivel de Logro:* 100%. Completamente logrado, el código implementado ha compilado, se logra simular luego de la implementación, se logra generar bitstream y cargarlo en la ZYBOZ7.

2) Actividades Complementarias

AC1
- *Descripción:* Existen múltiples máquinas de estados a lo largo del proyecto, unas de las mas importantes son: la de la FFT, la cual se mueve entre 5 estados, activando diferentes procesos. Los 5 estados son: *IDLE*, *INIT*, *LOAD_COEF*, *PROCESSING* y *DONE*. Y la máquina de estados de Vitis, la cual maneja el estado de la pantalla, lo cual afecta al funcionamiento de interrupciones, funciones y el uso de variables.
- *Nivel de Logro:* 100%. Completamente logrado, el código implementado ha compilado, se logra simular luego de la implementación, se logra generar bitstream y cargarlo en la ZYBOZ7.

AC2
- *Descripción:* Se utiliza todo lo solicitado, algunos ejemplos son:

Estructuras: 'ReadResult' en el archivo sdFunctions.c, el cual se usa para recibir un puntero que lleva a 64 datos de la SD cada vez, que serán enviados a la FFT.

Arreglos: 'song_titles' definido en globals.c, es un arreglo de arreglos con los diferentes títulos de las canciones según cada *mood*.

Punteros: Al igual que el resto de lo solicitado, se utilizan para muchos casos, en particular en la función update_day_night() del 'utils.c', se usan punteros para los datos día y prev_dia. Esto es porque los valores también se usan en otros archivos por lo que facilita su operación.

Funciones: Se usan funciones a lo largo de todo el proyecto, las mas evidentes son los que manejan el estado actual llamados estado**X**_handler() con **X** entre 0 y 3.

Macros: En 'globals.h' podemos encontrar múltiples macros, por ejemplo al tratar con *BACKGROUND*, *FOREGROUND*, los pines

para el GPIO, la cantidad de naciones (NUM_SONGS), etc.

- *Nivel de Logro:* 100%. Completamente logrado, el código implementado ha compilado, se logra simular luego de la implementación, se logra generar bitstream y cargarlo en la ZYBOZ7.

AC3
- *Descripción:* Como periféricos adicionales utilizamos el Joystick, una de las perillas y el sensor de temperatura.
- *Nivel de Logro:* 100%. Completamente logrado, el código implementado ha compilado, se logra simular luego de la implementación, se logra generar bitstream y cargarlo en la ZYBOZ7.

AC4
- *Descripción:* El *Boot* se implementa correctamente, permitiéndonos acceder al proyecto sin necesidad de reprogramar la Zybo.
- *Nivel de Logro:* 100%. Completamente logrado, el código implementado ha compilado, se logra simular luego de la implementación, se logra generar bitstream y cargarlo en la ZYBOZ7.

AC5
- *Descripción:* Se implementa ILA para analizar las transacciones de AXI, en particular para ver los datos enviados desde la Zynq hacia el IP-Core 'AXIFloat' (creado por nosotros). Además, usamos VIO para poder ver los cambios de múltiples variables diferentes, una de las más útiles es la salida del gpio que se conecta con el IP-Core 'Buzzer'. La diferencia entre VIO e ILA radica en que la primera nos permite observar los valores seleccionados en tiempo real mientras que el otro los captura en una ventana de tiempo. Si bien ambas nos son de mucha utilidad para *debuggear* y entender el comportamiento de nuestro proyecto, estas tienen objetivos diferentes. En nuestro caso, aquella que nos fue de mayor utilidad fue la VIO.
- *Nivel de Logro:* 100%. Completamente logrado, el código implementado ha compilado, se logra simular luego de la implementación, se logra generar bitstream y cargarlo en la ZYBOZ7.

AC6
- *Descripción:* Se implementa la música en el Buzzer gracias a lo analizado por la FFT, si bien este sonido no es idéntico a lo que debiese reproducirse, esto es porque el análisis de frecuencia tarda más de lo esperado y también por que la reproducción depende del uso de *timers*. Por otra parte, una canción posee múltiples notas en simultáneo, y, por el funcionamiento del *buzzer*, podemos enviar una cantidad limitada de notas por instante por lo que simplemente se muestra la más relevante. A pesar de ello, consideramos que esto cumple con lo solicitado.
- *Nivel de Logro:* 100%. Completamente logrado, el código implementado ha compilado, se logra simular luego de la implementación, se logra generar bitstream y cargarlo en la ZYBOZ7.

AC7
- *Descripción:* Se logra utilizar la tarjeta SD correctamente, permitiéndonos obtener las diferentes canciones que solicitemos.
- *Nivel de Logro:* 100%. Completamente logrado, el código implementado ha compilado, se logra simular luego de la implementación, se logra generar bitstream y cargarlo en la ZYBOZ7.

III. RESULTADOS (3 PUNTOS)

A continuación se detallan las observaciones y resultados obtenidos mediante el uso de herramientas de *debug* en Vitis.

Inicialmente, se tomaron imágenes del VIO en su estado inicial, donde todas las variables se encontraban en 0. A continuación, se capturó una imagen del VIO con el proyecto en pleno funcionamiento, donde se pudo observar la variación de todas las variables, excepto `POT1`, el cual se mantenía en 0. Finalmente, se registró una imagen similar pero con `POT1` al máximo. Estas diferencias son notables tanto en el valor de entrada del `POT1` como en la palabra de control del `GPIO Out`.

Dado el tamaño de las imágenes, el cual dificulta su visualización, se adjuntan en una carpeta dentro del proyecto.

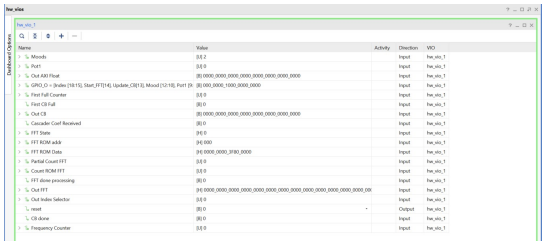

FIGURE 5: VIO en estado inicial.

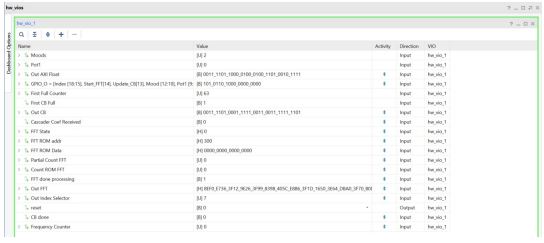

FIGURE 6: VIO con el proyecto en funcionamiento y `POT1` en 0.

Posteriormente, se procedió a la depuración del código en Vitis, colocando un punto de interrupción en el temporizador encargado de enviar el buffer de 64 datos en formato FP32 hacia el bloque AXI Float, el cual a su vez lo pasará al Circular Buffer (CB). Este proceso continuará incrementando el número de datos almacenados hasta alcanzar 64, momento en el cual se levantará la bandera indicando que el buffer está lleno.

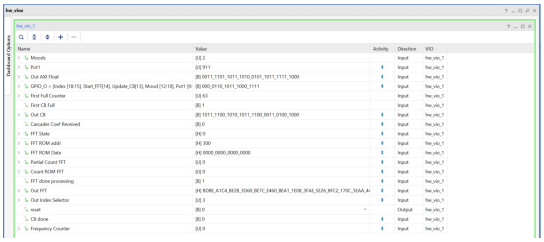

FIGURE 7: VIO con el proyecto en funcionamiento y `POT1` al máximo.

Se incluyeron dos imágenes adicionales para ilustrar este proceso: una de Vitis y otra del VIO. En estas, se puede observar cómo el estado del AXI Float cambia respecto al estado inicial y cómo aumenta el contador del CB. Finalmente, se observa la activación de la bandera de llenado del buffer. Es importante destacar que se presenta una diferencia entre el contador de Vitis y el del hardware, lo cual se debe a que el CB ignora los valores consecutivos iguales, resultando en un tiempo mayor para llenarse. No obstante, el código está diseñado teniendo en cuenta esta particularidad.

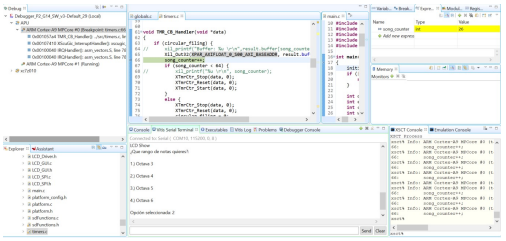

FIGURE 8: Depuración en Vitis con punto de interrupción en el temporizador.

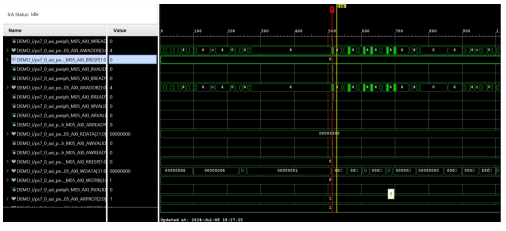

FIGURE 9: ILA mostrando comunicación AXI

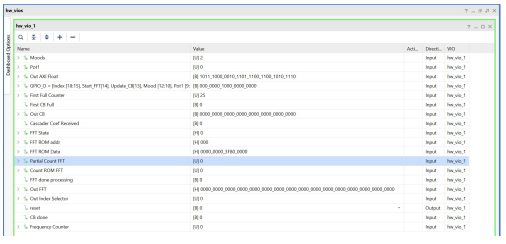

FIGURE 10: Estado del AXI Float y contador del CB en VIO.

En conclusión, las herramientas de depuración en Vitis, junto con los análisis proporcionados por ILA y VIO, permitieron identificar y destacar las diferencias clave en el comportamiento del proyecto, especialmente en lo que respecta a la variación de `POT1` y la gestión del *buffer* en el Circular Buffer.

IV. RESULTADOS IMPLEMENTACIÓN (0.1 PUNTOS)

Los principales problemas encontrados con la implementación fueron a la hora de unir las diferentes partes del proyecto, intentando mantener el orden de los archivos y sin afectar negativamente al resto del funcionamiento. Algunos casos puntuales fueron los *timers* y su sincronización con la FFT y la implementación de la SD. Sin embargo, logramos superar todos estos problemas correctamente.

V. CONCLUSIONES (0.2 PUNTOS)

- Hubo una diferencia en el tiempo de ejecución de 2 segundos respecto a la canción original. Esto se debe a que la canción se muestrea a 44.1 kHz y no es posible dividir 100 MHz por un número entero que encaje perfectamente en esta frecuencia, lo que generó un redondeo y, en consecuencia, esta diferencia.
- Dado que el Buzzer solo puede tocar una nota a la vez, la fineza y los armónicos de la canción no se aprecian correctamente, especialmente porque nos centramos en una sola octava. Debido a esto, muchas veces solo se recupera una idea del ritmo y los cambios en la estructura musical de la canción original.
- Para simplificar y evitar problemas de sincronización, decidimos que todo el hardware operara con las diferencias entre datos actuales y anteriores. Esto introduce un error evidente: si la canción tiene dos veces el mismo dato, este es ignorado, creando saltos en la canción. Esto también contribuye a las diferencias percibidas entre la canción real y la que toca el Buzzer.
- Otro problema es que cada vez que tocamos la canción existen leves diferencias. Esto se debe a que esperábamos siempre tener el mismo input en el bloque de la FFT, pero por pequeños problemas de sincronización, algunos datos de la canción anterior permanecen, desfasando la canción actual. Este desfase suele ser pequeño (1 o 2 muestras), pero es apreciable al oído dado que se toma la FFT de 64 muestras.
- Debido a problemas de sincronización, si se detiene la canción en medio de una petición de FFT y no se alcanza a mandar el *flag* de cambio de datos a la salida del Circular Buffer, la FFT se ejecuta sin cambios en los datos de entrada, lo cual la bloquea dado el funcionamiento por examinación de cambios en las entradas. Esto ocurrió muy pocas veces; en casi 60 cambios de canciones, ocurrió una sola vez.

En general nos percatamos que queda trabajo por hacer para robustecer su funcionamiento a esas pequeñas condiciones que no fueron previstas. En particular los problemas

de sincronización fueron los mas complejos de identificar y requirieron de una análisis mas fino.

VI. TRABAJOS FUTUROS (0.2 PUNTOS)

Posibles mejoras a realizar en nuestro proyecto podrían ser:

1) **Refactorizar el código** para poder evitar *hardcodeos* y hacer el código más pulcro y limpio considerando buenas prácticas de programación en C.

2) **Mejorar la FFT y el circular Buffer**, en términos de eficiencia, permitiendo tener un código depurado y de mejor calidad evitando que esto sea un cuello de botella para el buzzer.

3) Hubiese sido interesante poder guardar la lectura del micrófono para después **Reproducir la voz** como lo hacemos con las canciones.

4) Otra mejora posible sería la **limpieza del** '***Circular Buffer***' cuando se cambia de canción, la falta de este procedimiento provoca que las canciones suenen siempre diferentes al partir ya que, por detrás, se está analizando la información anterior mas el comienzo de la canción seleccionada.

REFERENCES

[1] OpenAI, *Generative Pre-trained Transformer 4 (GPT-4)*, OpenAI, 2023, URL https://www.chatgpt.com/.

[2] Anthropic, *Claude 3.5 Sonnet*, Anthropic, 2024, URL https://claude.ai/new.

[3] A. Kapitanov, "fp32 logic," in *GitHub Repository*, 2018, commit. 7a0a6937496c16d1f6a6b7b5e1b0d93b65078825, URL https://github.com/hukenovs/fp32_logic.