Fecha de entrega 11 de mayo de 2024.

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

Identificador de notas

ETIENNE CABIAC[1], MATEO DE LA CUADRA[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 un analizador de sonidos tanto en frecuencia como en magnitud. Para ello se debe cargar en la rom del player un archivo .coe que contenga los snidos en forma de bits. Luego, el análisis de frecuencia nos permite encontrar la frecuencia de los 64 primeros sonidos y mostrar un color asociado a esta en el led RGB, por otra parte la magnitud se muestra constantemente en los leds normales, encendiendo entre 0 y 4 leds. Para llevar a cabo el proyecto utilizamos Vivado junto con una Zybo Z7-10, en particular, utilizamos VHDL como lenguaje de prograación. En términos de conocimientos importantes, tuvimos que manejar señales, implementar una FFT para el análisis de frecuencia y manejar AXI. Además utilizamos múltiples conceptos y convenciones, como floating point (IEEE754), PWM, etc. En términos de resultados, logramos llevar a cabo lo planteado pero existen mejoras posibles por aplicar.

INDEX TERMS VHDL, FFT, AXI, sonido.

I. ARQUITECTURA DE HARDWARE (1 PUNTO)

EN términos de arquitectura de hardware, proyecto tiene 3 áreas principales: El reproductor de sonido (en forma de bits), el análisis de frecuencia y el análisis de magnitud de la señal recibida.

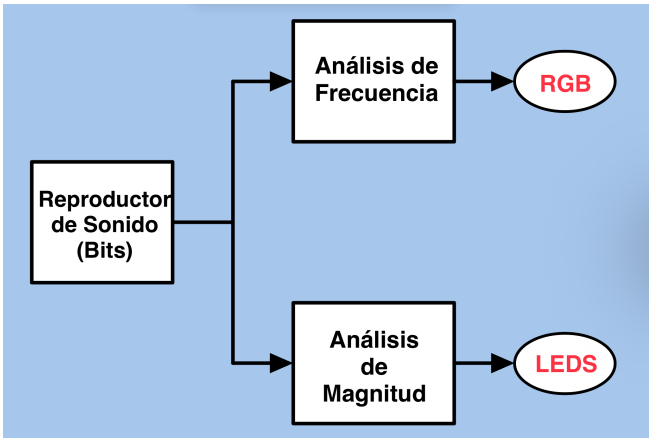

FIGURE 1: Esquema general del proyecto.

Cada una de estas áreas incluye múltiples bloques que actúan en conjunto para llegar al abjetivo deseado, el cual es, por una parte, representar las frecuencias de una canción a lo largo del tiempo, con diferentes colores en el led RGB. Y por otra parte, analizar la magnitud de las señales, representando por tanto la intensidad del sonido en los leds, encendiendo entre 0 y 4 leds.

A. REPRODUCTOR DE SONIDO

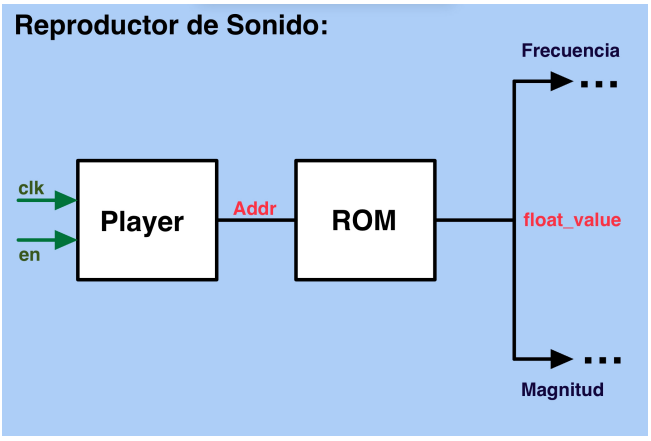

FIGURE 2: Reproductor de sonido.

Dentro del reproductor de sonido, podemos encontrar 2 módulos, "*Player*" y una ROM de sólo lectura que contiene un archivo '.coe' con los bits del sonido a mostrar (Este se obtiene utilizando el archivo 'sampler cancion.py' ubicado en la carpeta '_utils').

El primero, contiene un valor genérico '*MAX_COUNT*', que establece la velocidad del clock interno, con el cual cambia el valor de la señal '*address*' y por tanto, el valor que retorna la ROM. Por su parte, la ROM retorna el valor almacenado en el *address* denominado '*float_value*' y se lo entrega a los bloques "*circular_buffer*" y "*amplitude_converter*" de las áreas de análisis de frecuencia y de magnitud respectivamente.

B. ANÁLISIS DE FRECUENCIA

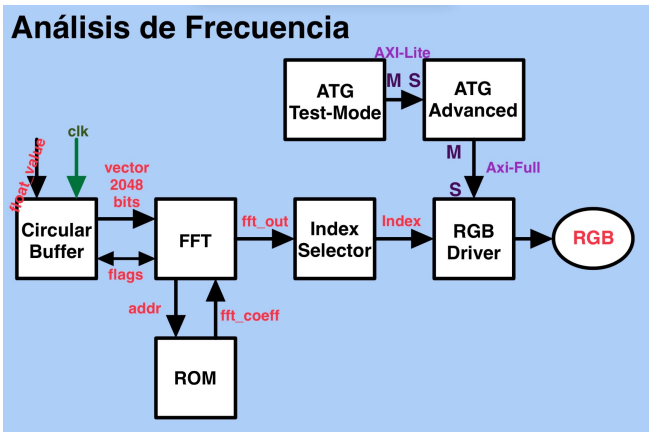

FIGURE 3: Esquema del análisis de frecuencia.

La idea general del análisis de frecuencia es ir recibiendo una serie de valores de sonido, pasarlos por una FFT, la cual analiza las frecuencias y quedarnos con aquella de mayor relevancia. Una vez obtenida dicho indice (y por tanto su frecuencia), mostramos su color asociado en el led RGB.

En un principio utilizaríamos AXI pero no tuvimos tiempo de juntar la parte de AXI advanced con lo anterior.

1) Circular Buffer

El bloque *circular_buffer* es el encargado de recibir la señal 'float_value' del reproductor, e ir concatenando su valor hasta tener un gran vector de 2048 bits. Una vez que se obtiene dicho vector, se lo entrega al bloque de la FFT, junto a una flag *is_done* que permite que la FFT se ejecute. Es importante mencionar que este módulo también recibe una flag de parte de la FFT (*FFT_done*), esto es para evitar errores internos dentro de la FFT. Sin embargo, la primera vez que el *circular_buffer* se llena (le llega el vector n° 64) activa la *is_done* para que se inicie por primera vez el sistema.

2) FFT + ROM

La FFT funciona recibiendo un vector de 2048 bits (64 vectores de 32 bits concatenados) y una flag de partida por parte del módulo *circular_buffer*. Internamente, la FFT suma y multiplica múltiples valores, provenientes de los la ROM, en cascada, obteniendo 12 vectores con distintos valores, los cuales representan 12 frecuencias diferentes y son exportados por el bloque.

3) Index Selector

Este bloque tiene por objetivo identificar, a partir de los 12 valores obtenidos por el módulo anterior, aquél de mayor valor, entregando un index el cual se relaciona directamente con la frecuencia.

4) RGB Driver + ATG's

Por último, el *rgb_driver* recibe a través de AXI-Full (ATG en 'Advanced Mode') los valores de encendido para los leds RGB, por su parte dicho ATG obtiene los valores por *AXI-Lite* (otro ATG, en 'Test Mode'). Luego, según el valor del index recibido, le asigna el valor correspondiente (obtenido por *AXI-Full*) a los led RGB.

C. ANÁLISIS DE MAGNITUD

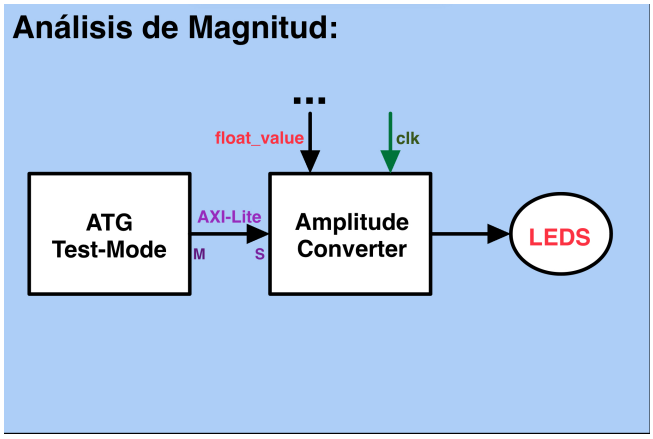

FIGURE 4: Esquema del análisis de magnitud.

El análisis de magnitud se resume en el bloque *'amplitude_converter'* el cual recibe *float_value* por parte del reproductor y además, recibe, a través de *AXI-Lite* los valores internos con los que compara *float_value* para determinar la intensidad del sonido actual y por tanto encender entre 0 y 4 leds.

II. ACTIVIDADES REALIZADAS (1.5 PUNTOS)

1) Actividades Obligatorias

AO1
- *Descripción:* El código cumple con utilizar *components*, integrando más de tres componentes dentro de otra entidad. En particular, en el bloque FFT de la sección I-B. El uso de las entidades permiten, sumar en cascada (CU Cascader), multiplicar en cascada (FPU Cascader) y cambiar el formato de los datos (Std To Fp32).
- *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:* Se crearon y utilizaron múltiples *IP-Cores*, en particular, algunos de estos que utilizan parametros genéricos son: *player*, *amplitude_converter* y también el *RGB_driver*. Dichos

parámetros se usan principalmente para: determinar la rapidez del reproductor, settear los parámetros de AXI y para establecer el contador del PWM. Ver I.

- *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:* Creamos e instanciamos 2 *IP-Core* comunicados por *AXI*. Estos son, *RGB Driver*, conectado a un *ATG* maestro en *Advance Mode*, y *Amplitude Converter* conectado a un *ATG* maestro en *Test Mode*.
- *Nivel de Logro:* 83.3%. Ambos bloques han sido creados, simulados y es posible generar su bitstream, sin emabrgo, el *RGB Driver* no se alcanzó a implementar en el código Main.

2) Actividades Complementarias

AC1
- *Descripción:* Existen múltiples máquinsa de estados a lo largo del proyecto, una de las mas importantes es 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*.
- *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 Vio e ILA para monitorear y modificar señales, de hecho, con el fin de poder entender lo que ocurre de forma mas sencilla, dejamos como outputs múltiples variables extras.
- *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:* A lo largo del proyecto usamos múltiples operadores como sumas, restas, multiplicaciones, y operadores lógicos (AND, OR...) así como distintos atributos, por ejemplo, variable'HIGH y variable'EVENT. Para las operaciones, podemos fijarnos en los bloques internos de la FFT (FPU Cascader y Cu Cascader), y, para los eventos podemos verlos en el *index_selector*.
- *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:* Se implementan variables en el proyecto, en particular en el *Circular_Buffer* implementa 2 variables los cuales sirven como contadores y son actualizados de manera instantánea. Un ejemplo es la variable 'done_counter'.
- *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 usa código secuencial así como combinacional en diferentes momentos, hay módulos con ambos tipos usados y algunos que son exclusivamente de un tipo. Entre ellos, se encuentran: *index_selector* (Mixto), *st2_to_fp2* (Combinacional), *circular_buffer* (Secuencial). La principal diferencia es la instantaneidad de ejecución, es decir, uno espera que ocurra algo para ejecutarse (secuencial) mientras que el otro ocurre constantemente (combinacional).
- *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:* En el archivo *index_selector* podemos encontrar una función y una procedure. La diferencia radica en si es que el subprograma puede no retornar una salida. La diferencia de uso radica en que podemos no tener una salida del array vectors, pero no un podemos no tener un index.
- *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:* Las funciones que implementamos que no han sido presentadas en el curso son: ROM's y la FFT (la cual terminamos creando nosotros mismos).
- *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 DE SIMULACIÓN (3 PUNTOS)

Muestre su simulación post implementación de alguno de sus procesos con el fin de:

- (0.75puntos) Identificar los retardos (delays) que ocurren debido a la implementación real y que no existen en la simulación de comportamiento.
  Para analizar este item, presentamos las simulaciones post-implementación y la comparamos con la simulación de comportamiento del modulo amplitude_converter. Aquí se hace evidente como el valor de las señales *leds* se demoran un tiempo más en responder cuando existe implementación (parte inferior de las imagen) versus cuando no existe (parte superior de las imagen).
  En esta imagen podemos ver como tarda más del triple en actualizarse.
  Y en esta otra, como el *delay* es de tan solo 0.1 ns.

- (0.75puntos) La diferencia entre la actualización de señales y variables.

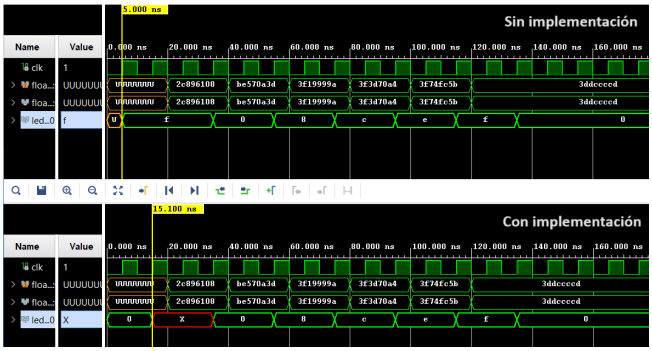

FIGURE 5: Comparación 1 de síntesis

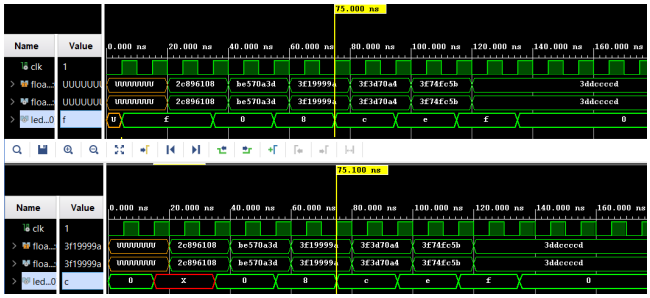

FIGURE 6: Comparación 2 de síntesis

Corriendo la simulación del siguiente código se puede demostrar lo solicitado:
Como podemos ver, la variable es actualizada en el mismo instante en el que se lo pedimos mientras que la señal se actualiza en el proximo ciclo de *clock* disponible.

```
architecture Behavioral of varsig is
    signal count: integer := 0;
begin
    test: process(clk) is
    variable cont : integer := 0;
    begin
        if rising_edge(clk) then
            cont := cont + 1;
            count <= count + 1;
            if count > 5 then
                clk_out_sig <= '1';
            end if;
            if cont > 5 then
                clk_out_var <= '1';
            end if;
        end if;
    end process;

end Behavioral;
```

FIGURE 7: Código señales vs variables

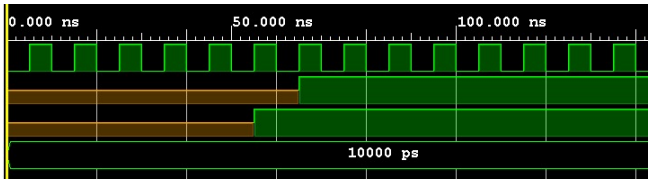

FIGURE 8: Diferencia de actualización entre señales y variables

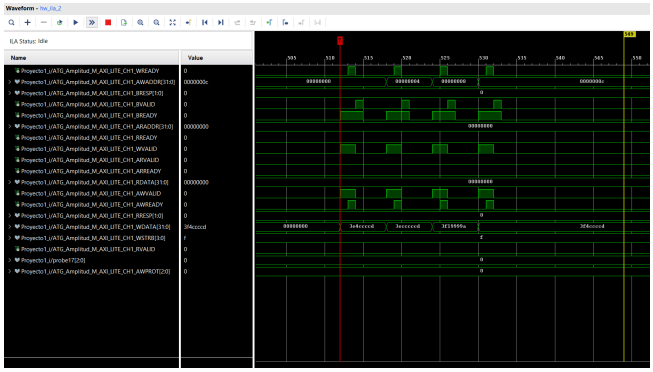

FIGURE 9: Comportamiento de AXI-Lite según ILA

- (2puntos) Mostrar la correcta ejecución de las transacciones AXI Lite y AXI Full. Utilice Testbench o ILA. Identifique claramente los handshake asociados a cada transaccion.
  Utilizando *ILA* en modo *AXI*, podemos ver el siguiente comportamiento, teniendo como *trigger* el valor de AWVALID.
  Para el caso de AXI-Full, tenemos 2 *ILA's*, una entre la conexión de los *ATG's* y otra para la conexión de AXI-Advanced.

IV. RESULTADOS IMPLEMENTACIÓN (0.1 PUNTOS)

Los principales problemas encontrados con la implementación fueron a la hora de juntar todo, ya que los bloques funcionaban perfectamente por separado, pero al manejar variables que cambiaban de manera incontrolable era imposible de predecir el comportamiento. Además, para probar cosas nuevas, era muy tedioso tener que esperar la generación de un nuevo bitstream que podía llegar a tardar más de 15 minutos. Por último, optamos por cambiar el objetivo del

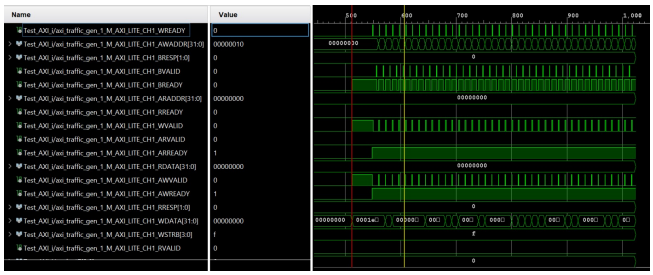

FIGURE 10: Transacción entre ATG's

proyecto en el área de frecuencia I-B

V. CONCLUSIONES(0.2)

- Por problemas de sincronismo, la FFT solo era capaz de correr una vez, en lugar de correr cíclicamente como se habia simulado en un comienzo.
- Nos percatamos de la importancia y versatilidad que ofrecen las maquinas de estados para mitigar los problemas de sincronizacion y latencia.

VI. TRABAJOS FUTUROS (0.2)

Algunas mejoras posibles para el proyecto son:

1) **Sintonizar las señales de clock**, esto es, implementar una señal que permita que el reproductor emule, en tiempo real, la canción, permitiendo al ususario tener una experiencia envolvente, mostrando la frecuencia e intensidad acorde a cada instante.

2) **Mejorar la FFT**, en términos de eficiencia, permitiendo tener un código depurado y de mejor calidad. Tambien permitir que el bloque pueda correr mas de una vez sin problemas.

3) En caso de poder implementar **periféricos**, sería interesante poder transmitirle la señal de sonido a una bocina, lo cual a su vez cumpliría parcialmente con el primer punto.

REFERENCES

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

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