UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA
FACULTAD DE CIENCIAS QUÍMICAS E INGENIERÍA

Programa de Ingeniería en Electrónica

Tesis

SINTETIZADOR MUSICAL POR MODULACIÓN DE FRECUENCIA IMPLEMENTADO EN UN FPGA

Que para obtener el título de Ingeniero en Electrónica

Presenta

MARIO ALFREDO MONTOYA SANDOVAL

Tijuana, B.C. **Abril de 2016**

Universidad Autónoma de Baja California
FACULTAD DE CIENCIAS QUÍMICAS E INGENIERÍA

FOLIO No. 002

Tijuana, B. C., a 18 de abril de 2016

C. Mario Alfredo Montoya Sandoval
Pasante de: Ingeniero en Electrónica
Presente

El tema de trabajo y/o tesis para su examen profesional, en la Opción TESIS

Es propuesto, por el C. Dr. Eduardo Álvarez Guzmán

Quien será el responsable de la calidad de trabajo que usted presente, referido al tema "Sintetizador musical por modulación de frecuencia implementado en un FPGA"

el cual deberá usted desarrollar, de acuerdo con el siguiente orden:

I.- INTRODUCCIÓN
II.- ANTECEDENTES Y MARCO TEORICO
III.- MÉTODO PROPUESTO
IV.- IMPLEMENTACIÓN
V.- PRUEBAS Y RESULTADOS
VI.- CONCLUSIONES Y TRABAJO FUTURO

Dr. Eduardo Álvarez Guzmán
Presidente de Tesis

Dr. Luis Enrique Palafox Maestre
Director

Dr. José Luis González Vázquez
Subdirector

Resumen

Se presenta el diseño de un sintetizador con aplicación musical capaz de generar diversos timbres utilizando modulación de frecuencia (o, técnicamente, modulación de fase), implementado en un arreglo de compuertas programable en campo (FPGA por sus siglas en inglés).

En este trabajo, se integrarán los conocimientos obtenidos a lo largo de la carrera de Ingeniería en Electrónica, cursada en la Universidad Autónoma de Baja California campus Tijuana Otay, junto con conocimientos musicales adquiridos de manera personal.

En el marco teórico se presentarán los conceptos de teoría musical, física, matemáticas y electrónica digital que se requirieron tomar en cuenta para el diseño del sintetizador.

Además de comprobar su funcionamiento auditivamente, se llevó a cabo un experimento de validación en el cual se compararon las señales producidas contra señales simuladas utilizando MATLAB.

Agradecimientos

A mi Universidad, y especialmente a mis profesores (todos excelentes sin excepción) por la labor que realizan con tanta dedicación y calidad, que se refleja en lo que he logrado desarrollar en la presente tesis. Agradezco particularmente al Dr. Eduardo Álvarez Guzmán (asesor de tesis), a la Dra. Edith García Cárdenas, al M.C. José Jaime Esqueda Elizondo, al Dr. José Luis González Vázquez (docente de la materia de Dispositivos Reconfigurables FPGA), al I.E. Pedro Ayala Muñoz (por su demostración de una guitarra eléctrica conectada a un osciloscopio, un momento decisivo en mi carrera, y por motivarme a realizar la tesis), y al MBA Luis Manuel Luna Rosas.

A mis padres, a quienes prácticamente les debo el que haya logrado terminar tan exitosamente mi carrera, por las muchas formas de apoyo que recibí de ellos, desde el apoyo económico y el alimento, hasta los raites de emergencia y que me despertaran cuando me quedaba dormido.

A mis amigos, especialmente mi mejor amiga Saira Navarro, mi amiga y "gemela" Yvonne Méndez (Q.E.P.D.), y mi cómplice y siempre compañera de equipo durante la carrera Sandra Juárez.

Prólogo

El estudio físico-matemático de la música siempre me ha parecido algo hermoso, y quisiera transmitir este conocimiento al mundo de todas las formas posibles, con el fin de propagar la idea de que las ciencias y las matemáticas no son sólo números, pensamiento cuadrado, y resolver problemas sin aplicación en la vida cotidiana, sino que permiten entender el mundo en el que vivimos desde sus bloques más básicos hasta sus manifestaciones más complejas y majestuosas.

Por ello, aparte de definitivamente considerar la docencia en el futuro, desearía que el trabajo desarrollado en la presente tesis se aproveche para motivar la curiosidad y el asombro en los músicos, en los estudiantes de ingeniería de nuevo ingreso, y en las nuevas generaciones en general, especialmente considerando que la música es un tema con el que prácticamente cualquier persona se puede identificar de alguna manera.

Adicionalmente, cuento con un canal de Youtube localizado en:

www.youtube.com/AwesomeAudioChannel

donde periódicamente subo videos educativos relacionados con música y acústica.

Índice general

Índice de figuras

Índice de tablas

Glosario

ADC	Convertidor analógico-a-digital
BPM	Tiempos por minuto
CLB	Bloque lógico configurable
DAC	Convertidor digital-a-analógico
DDS	Sintetizador digital directo
DFT	Transformada discreta de Fourier
FFT	Transformada rápida de Fourier
FPGA	Arreglo de compuertas programable en campo
HDL	Lenguaje de descripción de hardware
IOB	Bloque de entrada y salida
JND	Diferencia apenas perceptible
LUT	Tabla de acceso
MIDI	Interfaz digital de instrumento musical
SPI	Interfaz de periférico serial
SPL	Nivel de presión sonora
VHDL	Lenguaje de descripción de hardware VHSIC
VHSIC	Circuito integrado de muy alta velocidad

Introducción

En las artes, existe una constante necesidad de encontrar nuevos conceptos y novedosas técnicas que le permitan al artista explotar su potencial creativo, que puede resultar simplemente de hacer más eficientes, versátiles y portátiles las herramientas previamente existentes, o de la creación y uso de nuevas herramientas que pongan a disposición del individuo formas de expresión que anteriormente no habían existido.

En el caso particular de la música, la generación de tonos musicales en épocas anteriores solía estar limitada a la utilización de dispositivos acústicos como cuerdas, cajas de resonancia y tubos. Sin embargo, hoy en día se cuenta con dispositivos electrónicos capaces de generar señales eléctricas, que posteriormente pueden ser transformadas a sonido a través de un altavoz.

Con esta alternativa, surge la posibilidad de producir nuevos sonidos que anteriormente hubieran sido imposibles de generar acústicamente debido a sus particulares formas de onda, además de poder modelar e imitar los sonidos de diversos instrumentos musicales sin la necesidad de contar físicamente con el instrumento, incluso utilizando un espacio físico reducido comparado con el del instrumento original.

Además, debido a que estos sonidos se sintetizan en el dominio eléctrico, es posible manipularlos con aún mayor detalle a través del uso de filtros electrónicos.

Los sintetizadores digitales son un ejemplo de tales dispositivos que introducen un nuevo espectro de posibilidades artísticas, y se han vuelto altamente relevantes en la escena musical desde su concepción, pues cumplen con las funciones descritas al inicio de esta introducción.

En el capítulo 1, se presenta brevemente la historia de los sintetizadores a través del tiempo, con el fin de ubicar en la línea del tiempo el tipo de sintetizador que se desarrollará. En el capítulo 2 se describen los fundamentos teóricos y matemáticos que se deben tomar en cuenta en el diseño de un sintetizador, tales como la física del sonido, la psicoacústica, y la electrónica digital. En el capítulo 3 se presenta el diseño de sintetizador propuesto, describiendo los parámetros y resultados deseados. En el capítulo 4 se redacta el desarrollo de la implementación, incluyendo la lógica que se siguió y los cálculos realizados. En el capítulo 5 se muestran resultados del diseño, los cuales son evaluados en el capítulo 6. En el capítulo 7 se establecen conclusiones derivadas del presente trabajo.

Con el fin de mostrar la evolución de los sintetizadores a través del tiempo y su gradual integración en la música, a continuación se presenta brevemente el origen de los sintetizadores y la secuencia de desarrollos que se llevaron a cabo respecto a los mismos.

1 Antecedentes históricos

El Telharmonium es un órgano electrónico inventado por Thaddeus Chill en 1897, que utilizaba un mecanismo de discos tonales rotando cerca de un imán y un inductor para generar tonos puros o llevar a cabo síntesis aditiva. El Telharmonium fue el primer instrumento en llevar este tipo de síntesis de forma electromecánica, aunque era impráctico por su tamaño, peso, y consumo eléctrico.

En el lapso entre 1915 y 1929 se desarrollaron varios instrumentos musicales electrónicos, entre ellos el Theremin, nombrado por su inventor Léon Theremin. Este particular instrumento contaba con dos antenas que detectaban la posición de las manos del usuario, para que éste pudiera controlar la frecuencia del tono musical con los movimientos en el aire de una mano, y la amplitud con la otra mano.

Los sintetizadores continuaron evolucionando, permitiendo nuevas formas de manipular el sonido. En los 30's y 40's se integraron en los sintetizadores elementos como osciladores, filtros y controladores de envolvente (control de amplitud en función del tiempo) para dar lugar a los sintetizadores sustractivos y los sintetizadores polifónicos, es decir, que pueden reproducir más de una nota musical al mismo tiempo. En la figura 1 se observa un órgano Hammond, construido por primera vez en 1935, que operaba con discos tonales al igual que el Telharmonium.

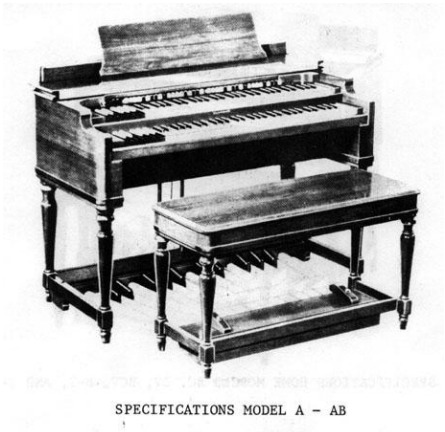


Figura 1: Órgano Hammond
(http://www.musicweb.ch/en/hammond-organ)

Figura 2: Sintetizador Minimoog
(http://www.vintagesynth.com/moog/voyager.php)

Durante los 40’s y 50’s comenzaron a aparecer los teclados electrónicos portátiles, siendo estos los precursores de los sintetizadores modernos. Su versatilidad al ser portátiles se veía contrastada por su limitante de ser monofónicos (únicamente podían reproducir una nota musical a la vez).

Los avances en la tecnología de estado sólido aportaron directamente a los sintetizadores de los 60’s y 70’s, volviéndolos más robustos al mismo tiempo que mantenían su portabilidad, por lo que comenzaron a ser adoptados en la escena musical, dando origen a géneros como el *synth pop*. Un ejemplo popular de un sintetizador de este periodo es el Minimoog, inventado por Robert Moog, mostrado en la figura 2.

En los 80’s se desarrolla el protocolo MIDI (*Musical Instrument Digital Interface*) facilitando la integración entre instrumentos electrónicos. El sintetizador Yamaha DX7 (figura 3) sale al mercado, y su gran éxito impulsa el uso y desarrollo de los sintetizadores digitales, dando inicio al decaimiento de los sintetizadores analógicos.

El sintetizador desarrollado en esta tesis está basado en el principio básico de modulación del Yamaha DX7.

En los 90’s aparecen los sintetizadores por software, facilitando la creación de sonidos y música a través de una computadora. En la actualidad, es común componer música a través de un secuenciador, que es un tipo de software en el cual se programan notas musicales a través de una interfaz gráfica, estableciendo sus alturas, timbres, duraciones, e intensidades, para posteriormente reproducir estas notas musicales de forma automatizada.

Figura 3: Sintetizador FM Yamaha DX7 (https://reverb.com/item/479871-yamaha-dx7-vintage-digital-polyphonic-fm-synthesizer)

2 Marco teórico

Para el desarrollo de un sintetizador desde sus bloques de funcionamiento más básicos, será necesario conocer los conceptos matemáticos y físicos involucrados en la síntesis de sonido, que se presentarán en esta sección.

2.1 Matemáticas de onda

Existe un conjunto de conceptos y herramientas matemáticas aplicables a todo fenómeno físico descrito como una vibración u oscilación, que serán elementales para el estudio del sonido debido a que este, como tal, es una onda.

Con el fin de comprender estos conceptos desde su base más fundamental, será necesario hablar primero de un tipo de movimiento mecánico específico, conocido como "movimiento armónico simple".

2.1.1 Movimiento armónico simple

Un oscilador simple está formado por una masa acoplada a un resorte, donde el movimiento de oscilación del resorte está restringido únicamente en dirección paralela al resorte, como se observa en el primer dibujo de la figura 4, etiquetado "reposo".

Si la masa se desplaza de su posición original y se suelta en el instante de tiempo t_0, y además se considera que no hubiera pérdidas que atenuaran la oscilación, el movimiento llevado a cabo por el sistema se denomina "movimiento armónico simple". En la misma figura 4 se observa una gráfica del movimiento realizado en el tiempo continuo, con los instantes t_0, t_1, t_2, t_3 y t_4 indicados en el eje de tiempo.

El movimiento armónico simple se puede describir en función del tiempo mediante la siguiente ecuación:

$$x = A\,cos(\omega_0 t + \Phi) \tag{1}$$

donde x es el desplazamiento con respecto a la posición en reposo en un instante específico de tiempo, A es la amplitud, ω_0 es la frecuencia angular (frecuencia de onda multiplicada por 2π), t es el tiempo, y Φ es la fase inicial.

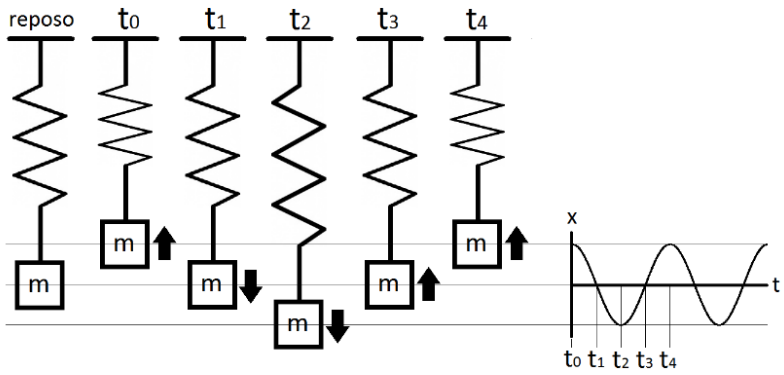


Figura 4: Oscilador simple en diferentes instantes de tiempo y representación gráfica de su movimiento

En la práctica, toda oscilación cuya gráfica tenga la forma observada en la figura 4, es referida como senoidal, independientemente de su fase de inicio. Esto incluye los casos descritos por un coseno, pues un coseno se puede considerar simplemente como un seno con un desfase de 90 grados.

En la figura 5, se muestra la medición de dos propiedades importantes de una oscilación senoidal: la amplitud, que se observa que es el máximo punto de oscilación y es medido desde el eje horizontal, y el periodo, que es el tiempo que toma una oscilación completa, antes de repetir el movimiento de forma idéntica.

La frecuencia de una onda, cuya unidad de medida son los Hertz (Hz, equivalente a seg^{-1}), es el número de ciclos completos que la onda realiza en un segundo. Teniendo el periodo de una onda, su frecuencia se puede calcular utilizando la siguiente ecuación:

$$f = \frac{1}{T} \tag{2}$$

donde f es la frecuencia de la onda en Hz y T es el periodo en segundos.

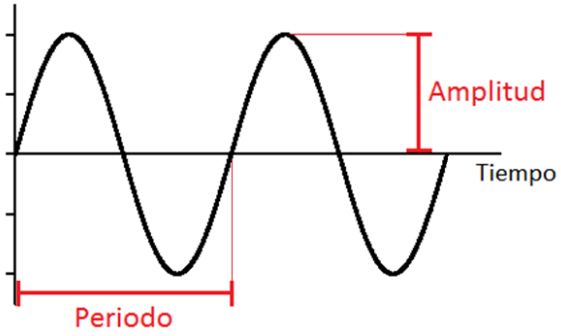


Figura 5: Visualización del periodo y la amplitud de una onda senoidal

2.1.2 Análisis espectral

El análisis espectral consiste en visualizar una señal variante en el tiempo en términos de una superposición de movimientos armónicos simples, es decir, senoidales, con diferentes frecuencias y amplitudes, y es indispensable en el análisis de señales variantes. Específicamente, en acústica nos permite inferir características de la percepción sonora de la onda, y predecir el efecto de filtros.

Una forma sencilla de comprender este concepto es revisando primero las series de Fourier, visualizando un ejemplo de las mismas, y posteriormente abordar el tema de la transformada de Fourier.

2.1.2.1 Series de Fourier

Las series de Fourier expresan que una señal periódica se puede describir en términos de una suma de senos y cosenos, como se muestra en la siguiente ecuación: [10]

$$x(t) = \frac{a_o}{2} + \sum_{n=1}^{N} [a_n \cos(2\pi n f t) + b_n sen(2\pi n f t)] \tag{3}$$

Donde $x(t)$ es la señal, a_o es el desplazamiento vertical (*offset*), n es el número de componente, a_n y b_n son la amplitud de cada componente, f es la frecuencia de la señal, y t es el tiempo.

Para ciertas formas de onda, como la cuadrada y la triangular, existen soluciones de la serie de Fourier en las que sólo es necesario introducir la amplitud y frecuencia deseadas para generar la onda matemáticamente. Por ejemplo, la serie de Fourier para generar una onda cuadrada es:

$$x_{cuadrada}(t) = \frac{4}{\pi} \sum_{k=1}^{\infty} \frac{sen(2\pi(2k-1)ft)}{(2k-1)} \tag{4}$$

$$= \frac{4}{\pi} \left(sen(2\pi f t) + \frac{1}{3} sen(6\pi f t) + \frac{1}{5} sen(10\pi f t) + \cdots \right) \tag{5}$$

La interpretación de la ecuación 5, entonces, es que una vez decidida la amplitud y la frecuencia deseada para la onda cuadrada, se comenzará con una senoide que tenga esa amplitud y frecuencia, a la que se le sumará otra senoide con 1/3 la amplitud y el triple de frecuencia, y a esa se le sumará otra senoide con

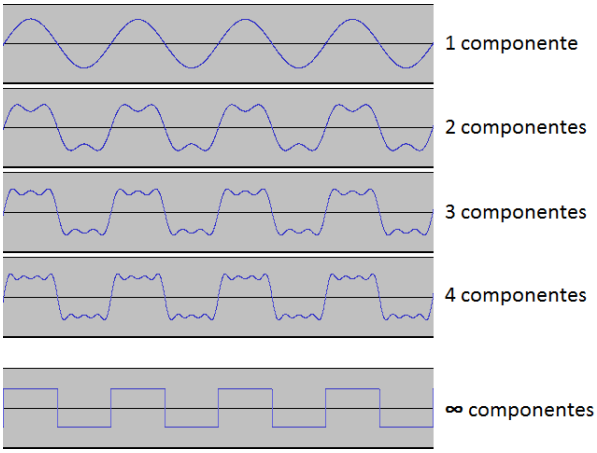


Figura 6: Representación visual de la serie de Fourier de una onda cuadrada

1/5 la amplitud y 5 veces la frecuencia original, etc., siguiendo este patrón. La amplitud final se obtendrá multiplicando la amplitud resultante por $4/\pi$.

En la figura 6 se muestra visualmente este procedimiento, sumando senoides una por una, y se observa que con cada suma consecutiva la forma de onda se aproxima más a la de una onda cuadrada. La serie es infinita, por lo que únicamente es necesario continuar con las sumas hasta que se alcance la precisión necesaria para la aplicación. Por ejemplo, en el caso de audio, sólo sería necesario sumar las componentes con una frecuencia menor a 20 kHz, ya que las frecuencias mayores estarían fuera del rango perceptible por el ser humano.

2.1.2.2 Transformada de Fourier

La transformada de Fourier, descrita como: [10]

$$G(f) = \int_{-\infty}^{\infty} g(t) e^{-j2\pi f t} dt, \tag{6}$$

transforma una señal en el tiempo $g(t)$, al espectro de frecuencias $G(f)$, que representa el conjunto de ondas senoidales, denominadas "componentes frecuenciales", que se tendrían que sumar en el tiempo para producir la señal analizada $g(t)$. Comúnmente, estas componentes son referidas simplemente como "frecuencias". Al graficar $G(f)$, se tendrá la frecuencia en Hz de cada componente en el eje horizontal (es decir, se está en el dominio de la frecuencia), y la amplitud en el eje vertical.

La utilización del exponencial *e* en la ecuación 6 es debido a que la identidad de Euler establece la siguiente relación:

$$e^{ix} = \cos x + i\text{sen}x, \tag{7}$$

cuyo propósito es obtener la amplitud correcta de la frecuencia que se está analizando en el momento, independientemente de su fase inicial, pues si la transformada incluyera sólo un seno o coseno, la amplitud correcta se obtendría únicamente comenzando el análisis en una fase en particular.

Si se aplicara la transformada de Fourier a una onda cuadrada, el espectro de frecuencia resultante sería el observado en la figura 7, que muestra las mismas frecuencias y amplitudes que se utilizaron en su serie de Fourier.

Entonces, considerando que cada componente frecuencial en un espectro corresponde a una onda senoidal, el espectro de frecuencia de una onda senoidal estaría formado por una sola componente cuya frecuencia y amplitud serían las mismas que las de la onda senoidal en el tiempo.

Por ejemplo, en la figura 8 se muestra una onda senoidal con una amplitud de 1.0 y un periodo de 10 ms. A partir de su periodo, se puede obtener su frecuencia utilizando la ecuación 2. Realizando el cálculo, se obtiene una frecuencia de 100 Hz. En el espectro mostrado en la misma figura 8 se muestra, efectivamente, una sola componente de 100 Hz con una amplitud de 1.0.

La señal en el tiempo se puede recuperar mediante la transformada inversa de Fourier:

$$g(t) = \int_{-\infty}^{\infty} G(f) e^{j2\pi f t} df \tag{8}$$

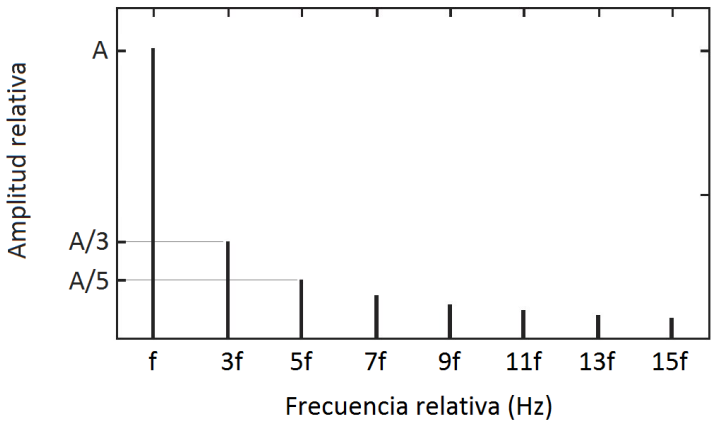
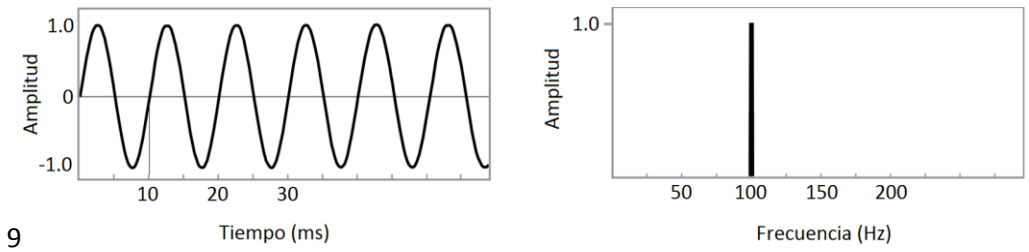


Figura 7: Espectro de frecuencia de una onda cuadrada



9

Figura 8: Onda senoidal en el dominio del tiempo (izquierda) y de la frecuencia (derecha)

2.1.2.2.1 Transformadas discreta y rápida de Fourier

Para señales discretas, es decir, que no están en función del tiempo continuo sino que están formadas por una cantidad finita de puntos, se utilizan las siguientes variantes de las ecuaciones 6 y 8, respectivamente, que corresponden a la transformada discreta de Fourier (DFT, *discrete Fourier transform*) y su inversa: [10]

$$G(k) = \left(\frac{1}{N}\right)\sum_{n=0}^{N-1} g(n)e^{-j2\pi kn/N} \tag{9}$$

$$g(n) = \sum_{k=0}^{N-1} G(k)e^{j2\pi kn/N} \tag{10}$$

donde $G(k)$ es el espectro discreto de frecuencias y $g(n)$ es la señal discreta.

A su vez, existe una variante de la transformada discreta denominada "transformada rápida de Fourier" (FFT, *fast Fourier transform*), que está optimizada para que se pueda calcular digitalmente la DFT con mucha mayor rapidez y eficiencia. Este es el método normalmente utilizado en software de audio.

Consiste en un algoritmo donde se realiza la DFT con un número de muestras igual a una potencia de 2. Con este algoritmo, el número de operaciones es de $N \log_2 N$, en vez de N^2 operaciones como era requerido originalmente en la DFT. Por ejemplo, para una transformada con 1,024 (2^{10}) muestras, la FFT lleva a cabo el cálculo 100 veces más rápido que la DFT [10].

De los diferentes tipos de modulación que existen, las modulaciones de frecuencia y fase son especialmente relevantes para el sintetizador desarrollado en la presente tesis, por lo que se describen a continuación.

2.1.3 Modulación de frecuencia y fase

La modulación de frecuencia (FM, *frequency modulation*) consiste en la variación de la frecuencia de una onda denominada "portadora", típicamente una senoide, en función de la amplitud instantánea de otra onda denominada "moduladora". La FM está definida por la siguiente ecuación: [11]

$$u(t) = A_c \cos[2\pi f_c t + 2\pi \Delta f \int_0^t x_m(\tau) d\tau], \tag{11}$$

donde $u(t)$ es la señal modulada, t es el tiempo, A_c es la amplitud de la portadora, f_c es la frecuencia de la portadora, Δf es la desviación de frecuencia (máxima diferencia de frecuencia respecto a f_c), y $x_m(\tau)$ es la onda moduladora, asumiendo que sus valores se encuentran entre -1 y 1.

Entiéndase "amplitud instantánea" como la amplitud de la señal en un instante de tiempo determinado, a diferencia de una amplitud *A* que representa la máxima amplitud de oscilación en el ciclo de la onda.

Para el caso específico de una moduladora senoidal, la modulación de frecuencia está dada por:

$$u(t) = A_c \cos\left[2\pi f_c t + \frac{\Delta f}{f_m} \sin(2\pi f_m t)\right], \tag{12}$$

donde f_m es la frecuencia de la onda moduladora. Al cociente $\frac{\Delta f}{f_m}$ también se le conoce como el índice de modulación β.

La modulación de fase (PM, *phase modulation*) consiste en variaciones de fase en la portadora de acuerdo a la amplitud instantánea de la moduladora, dada por:

$$u(t) = A_c \cos[2\pi f_c t + \Delta\theta\, x_m(t)]\,, \tag{13}$$

donde $\Delta\theta$ es la máxima variación de fase, y en este caso equivale por sí mismo al índice de modulación.

Ambos esquemas de modulación están relacionados debido a que una modulación de frecuencia con una cierta onda moduladora da el mismo resultado que una modulación de fase con la integral de esa

misma onda moduladora. En la figura 9 se muestran modulaciones FM y PM, donde se observa que una modulación FM con una onda cuadrada es equivalente a una modulación PM con una onda triangular.

Considerando que la integral de una senoide es simplemente otra senoide con diferente fase, modular con FM y PM utilizando para ambas una moduladora senoidal da el mismo resultado, únicamente desfasado, como se observa en la figura 10.

Las modulaciones FM y PM suelen categorizarse juntas bajo el nombre "modulación angular". La ecuación general de la modulación angular es:

$$u(t) = A_c \cos[2\pi f_c t + \Phi(t)]\,, \tag{14}$$

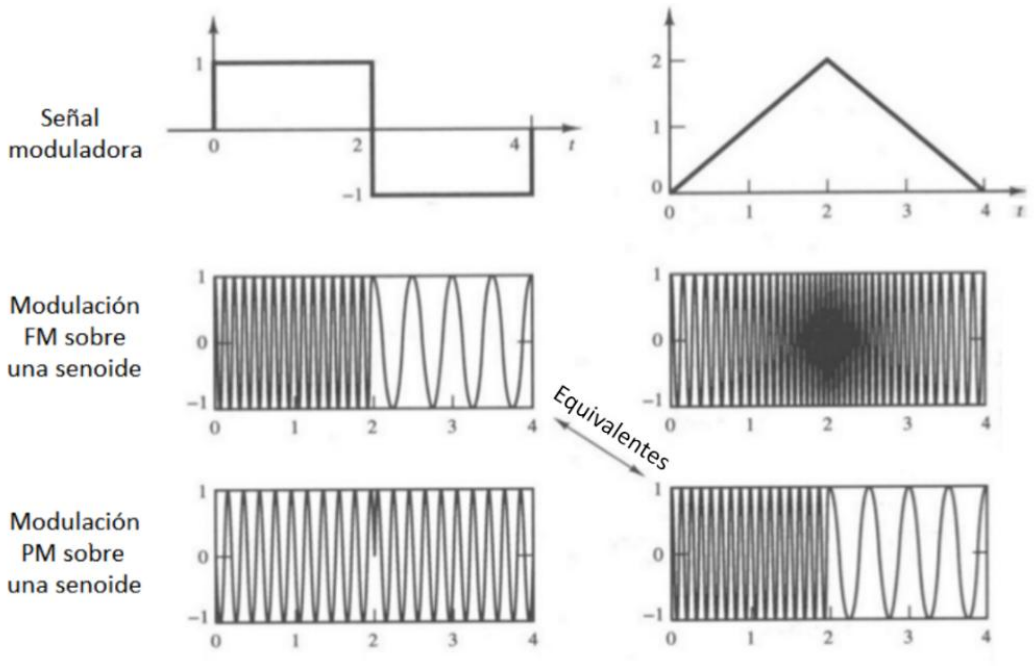


Figura 9: Ejemplos de modulación de frecuencia y de fase (Fundamentals of communications, Proakis y Salehi, p.332)

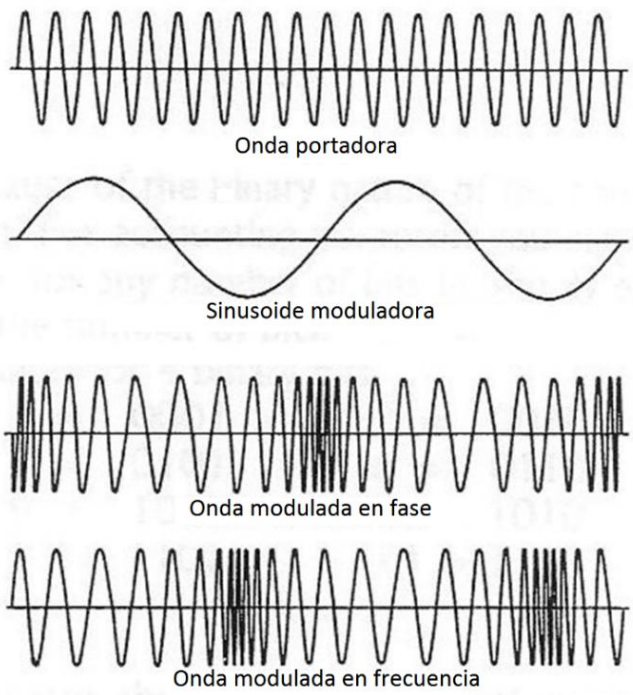


Figura 10: Modulación PM y FM con una moduladora senoidal (Communications Systems Design, Panter, p. 243)

2.2. Acústica

Una fuente de sonido, como un instrumento musical, se diseña tomando en cuenta el tipo de sonido que se desea que el oyente perciba, por lo que hay que considerar las características del sonido al realizar el diseño. Estas características y la percepción de sonido del oído humano se explicarán en esta sección.

2.2.1 Fenómeno físico del sonido

Definimos sonido como la percepción auditiva de una onda mecánica longitudinal de variaciones de presión, que viaja a través de un medio elástico. Aunque se puede tratar de prácticamente cualquier material, el medio más comúnmente tomado en cuenta en el estudio de sonido es el aire, ya que es el medio que usualmente está en contacto con el oído humano.

El sonido se origina por la vibración de un objeto. Al expandirse durante la vibración, el objeto empuja las partículas de aire frente a él (entiéndase "partícula" como un conjunto de moléculas de aire sólo lo suficientemente grande para que el promedio de sus movimientos aleatorios sea cero), comprimiendo el aire y aumentando la presión en esa región de aire.

Las partículas de aire en la región de alta presión tenderán a regresar a su estado de presión natural, y empujarán a las partículas adyacentes. Este fenómeno se repite, creando una onda de alta presión que se traslada. Hay que mantener en cuenta que las partículas como tal no se trasladan, sino únicamente vibran en su punto de equilibrio.

A continuación, cuando el objeto que vibra se contrae, jalará hacia él las partículas de aire frente a él, creando una región de baja presión. Al igual que en el caso de la región de alta presión, el aire en esa región tenderá a su estado natural, lo que jalará las siguientes partículas de aire, así dando origen a una onda propagante de baja presión.

Al repetirse el patrón de vibración de la fuente de sonido, continuamente se estarán propagando regiones de alta y baja presión, creando una onda de variación de presión, como se observa en la figura 11. Esta onda es finalmente percibida como un sonido al llegar a un receptor.

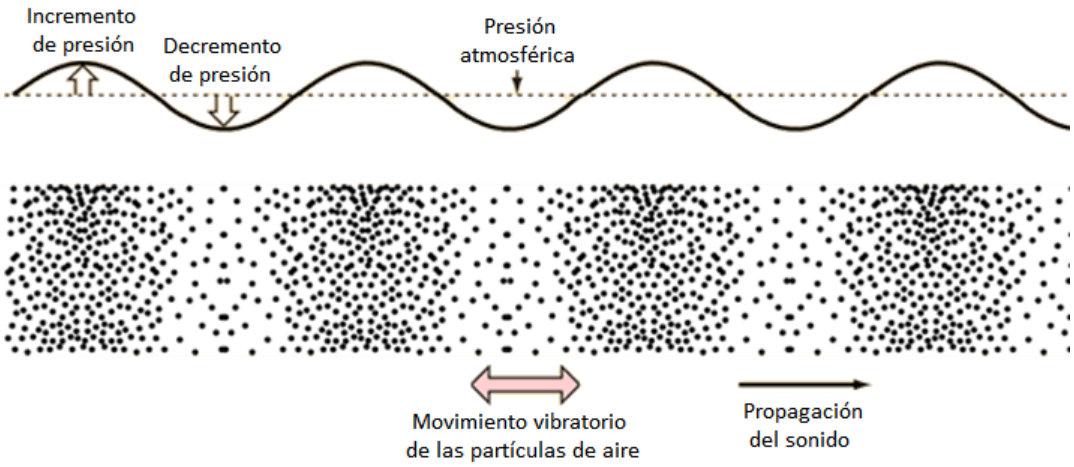


Figura 11: Representación gráfica de una onda de sonido propagándose por el aire (http://hyperphysics.phy-astr.gsu.edu/hbase/sound/tralon.html)

La velocidad de propagación del sonido dependerá de las condiciones del medio. Específicamente, la velocidad del sonido en aire seco a 20°C es de 343.2 m/s.

La frecuencia de una onda sonora, su velocidad, y su longitud de onda, están relacionados a través de la siguiente ecuación:

$$v = \lambda f \tag{15}$$

donde v es la velocidad de propagación de la onda en metros/segundo, λ es su longitud de onda en metros, y f es su frecuencia en Hertz.

2.2.2 Propiedades del sonido

Una onda sonora contará con 4 principales propiedades de las cuales dependerá el que el sonido se pueda utilizar bajo un contexto musical, y que se describirán a continuación.

2.2.2.1 Intensidad

De acuerdo con el diccionario de la Real Academia Española, éste concepto se define genéricamente como: Grado de fuerza con que se manifiesta un agente natural, una magnitud física, una cualidad, una expresión, etc.

En el contexto específico de la acústica, es la propiedad con la que distinguimos entre un sonido estruendoso y uno silencioso, y se le conoce coloquialmente como "volumen". Físicamente está dado por la magnitud de variación de presión Δp respecto a la presión atmosférica, causada por la distancia de desplazamiento de las partículas de aire durante su movimiento vibratorio. Esta variación de presión se mide en Pascales, dándonos su amplitud. Su representación gráfica se muestra en la figura 12.

Hay que tomar en cuenta que los términos "amplitud", "nivel", "intensidad", y "volumen" significan cosas ligeramente diferentes y se utilizan en diferentes contextos:

- "Amplitud" hace referencia a su máxima variación de presión en Pascales, medida en un punto físico del espacio.
- "Nivel" usualmente se utiliza al hablar de la magnitud sonora en escala logarítmica, representada en decibeles.
- "Intensidad" se refiere a la potencia sonora por unidad de área emitida por la fuente.
- "Volumen" es la magnitud percibida por una persona, la cual es subjetiva y no cuantificable por tratarse de una respuesta psicoacústica.

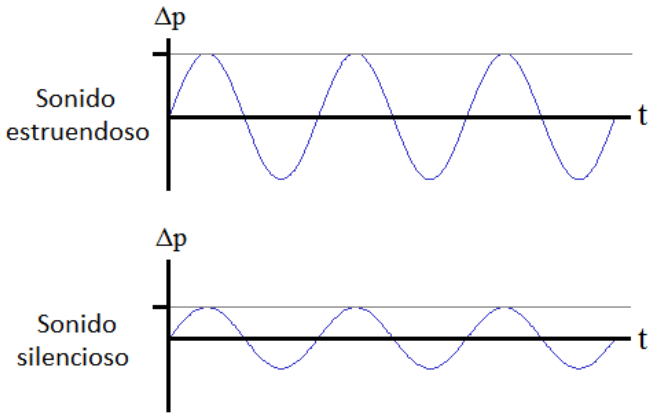


Figura 12: Par de ondas sonoras con diferente amplitud y sus percepciones correspondientes

2.2.2.1.1 Decibeles SPL

Debido al extenso de presionas sonoras audibles, que va de 10^{-5} Pa a 10^{2} Pa [22], es más común hablar de nivel de presión de sonido, abreviado L_p, cuya unidad son los decibeles SPL (*sound pressure level)*, y está dado por la siguiente fórmula:

$$L_p = 20 \log_{10}\left(\frac{p_{rms}}{20\mu Pa}\right) \tag{16}$$

donde p_{rms} es la media cuadrática de la variación de presión respecto a la presión atmosférica.

Los dB SPL utilizan como referencia 20µPa por ser el límite inferior de variación de presión que el ser humano es capaz de percibir, por lo que 0 dB SPL es el límite inferior de percepción de nivel de presión de sonido, y los decibeles negativos corresponden a niveles de presión de sonido imperceptibles. Se asume basado en pruebas psicoacústicas subjetivas que el oído humano percibe un incremento del doble de volumen con cada incremento aditivo de 10 dB.

El umbral del dolor, que es el nivel de presión de sonido que comienza a causar dolor en el oído humano, es de 130 dB SPL [16].

2.2.2.1.2 Decibeles FS

En software de audio, las amplitudes instantáneas de una onda en diferentes instantes de tiempo se expresan en un rango desde -1.0 a 1.0, como se muestra en la figura 13, independientemente de la resolución con la que haya sido digitalizada la onda.

Debido a este rango, no es apropiado expresar niveles de sonido en escala logarítmica utilizando decibeles SPL. En este caso, se requiere utilizar decibeles *full-scale* (dBFS) que están referenciados a la amplitud máxima en este rango, que sería 1.

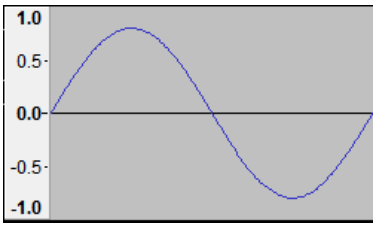


Figura 13: Escala utilizada en software de audio, mostrada en el extremo izquierdo (captura de pantalla de Audacity®)

Por lo tanto, el nivel de sonido en decibeles *full-scale* estaría definido por:

$$dBFS = 20\log_{10}\left(\frac{A}{1}\right) = 20\log_{10}(A) \tag{17}$$

donde A es la amplitud.

Considerando que todas las posibles amplitudes instantáneas serán menores a la referencia (o igual que la referencia, en caso de que la amplitud sea la máxima permitida), los niveles de sonido en dBFS serán decibeles negativos, o cero para el nivel máximo.

2.2.2.2 Duración

Es el lapso que dura una vibración audible. Generalmente se mide en segundos o milisegundos. En la figura 14 se muestran tonos telefónicos de timbrado y ocupado, con el tiempo en el eje horizontal y las unidades en segundos mostradas en la parte superior, y la amplitud en el eje vertical, donde se puede apreciar que los tonos de timbrado tienen una duración diferente que los de ocupado.

En música, la duración de las notas musicales se representa visualmente mediante figuras musicales, de las cuales la redonda tiene el valor de una unidad, y el resto tienen valores fraccionarios relativos a la redonda, como se muestra en la figura 15.

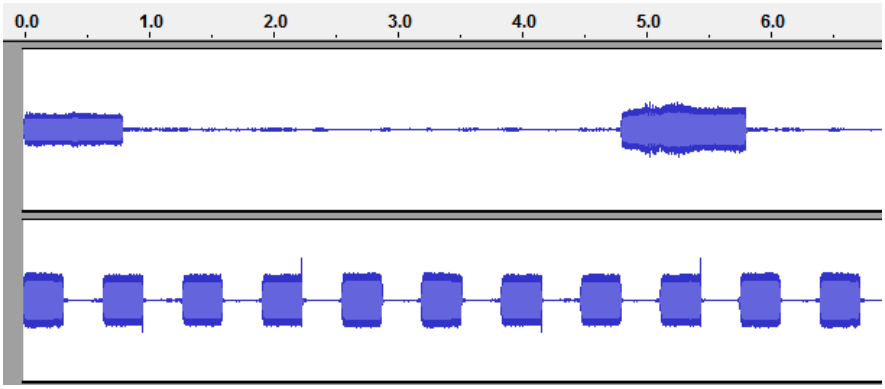


Figura 14: Tonos telefónicos de timbrado (arriba) y ocupado (abajo), grabados con el software Audacity®

Figura	Nombre	Valor	Tiempos (compás común)
	Redonda	1	4 tiempos
	Blanca	1/2	2 tiempos
	Negra	1/4	1 tiempo
	Corchea	1/8	1/2 tiempo
	Semicorchea	1/16	1/4 tiempo

Figura 15: Figuras musicales, sus valores, y su duración en 'tiempos' para el caso específico del compás común.

Una obra musical estará dividida en intervalos de tiempo iguales denominados "compases", como se muestra en la figura 16, y a su vez, estos estarán divididos en intervalos denominados "tiempos". En la figura 17, se muestran dos números 4 del lado izquierdo. De ellos, el primero indica que el compás tendrá una duración de 4 tiempos. El segundo establece que la figura musical cuyo valor es 1/4, es decir, la negra, es a la que se le asignará la duración de un tiempo. A este caso específico, de 4 tiempos equivalentes cada uno a una negra, se le conoce como "compás común", y la duración de las demás figuras musicales serían las mostradas en la última columna de la figura 15.

Alternativamente, una obra musical cuya partitura muestre los números 3 y 8, como en la figura 18, tendrá compases con una duración de 3 tiempos, y la figura musical cuyo valor es 1/8, es decir, la corchea, tendrá el valor de un tiempo. Por lo tanto la duración en tiempos de las figuras musicales sería diferente a las mostradas en la figura 15: la corchea sería equivalente a 1 tiempo, la negra a 2 tiempos, etc.

Cada obra musical cuenta con un parámetro denominado "tempo", expresado en tiempos por minuto (*BPM, beats per minute*) y especificada en una partitura como se muestra en la figura 19, que establece la rapidez con la que es ejecutada la obra musical.

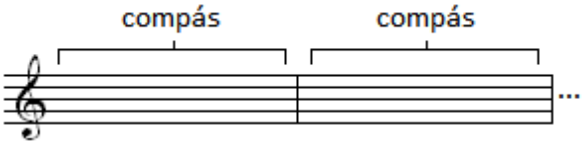


Figura 16: División de una pauta musical en compases

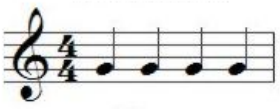


Figura 17: Compás 4/4 o compás común

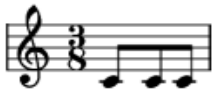


Figura 18: Compás 3/8

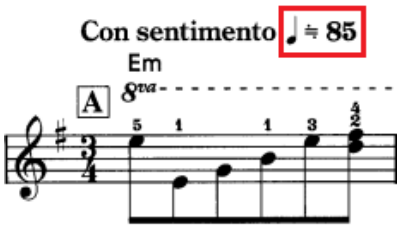


Figura 19: Localización del tempo en la partitura de "To Zanarkand" de Nobuo Uematsu (© 2001 Solid Co., Ltd. & Square Sounds Co., Ltd.)

La duración en segundos de una figura musical para una obra musical específica se calcula mediante la siguiente fórmula:

$$Duración\ de\ la\ figura\ [seg] = \left(\frac{duración\ de\ la\ figura\ en\ tiempos}{tempo\ de\ la\ obra}\right)\ x\ 60 \qquad (18)$$

Por ejemplo, considerando una obra con compás común, y un tempo de 140 tiempos por minuto, la duración de una negra se calcula de la siguiente forma:

$$Duración\ \text{♩} = \left(\frac{1\ tiempo}{140\ tiempos\ por\ minuto}\right) x\ 60 = 0.4286\ seg$$

2.2.2.3 Altura

Es la propiedad con la que se distingue un sonido grave de uno agudo, y es la característica que principalmente varía al tocar diferentes notas musicales en un mismo instrumento.

Está dada por la frecuencia del patrón completo de vibración de su forma de onda. A una nota grave le corresponde una frecuencia baja, mientras que a una nota aguda le corresponde una frecuencia alta, como se muestra en la figura 20.

Se le llama v*ibrato* al efecto musical producido una modulación de frecuencia lo suficientemente lenta para ser percibida por el oído como una variación ligera y repetitiva de su altura.

Una forma de obtener la frecuencia de un tono musical es primero identificando el patrón de oscilación y midiendo su periodo, es decir, la duración de un solo ciclo, como se muestra en la figura 21, y posteriormente calcular la frecuencia a través de la ecuación 2 (pág. 5).

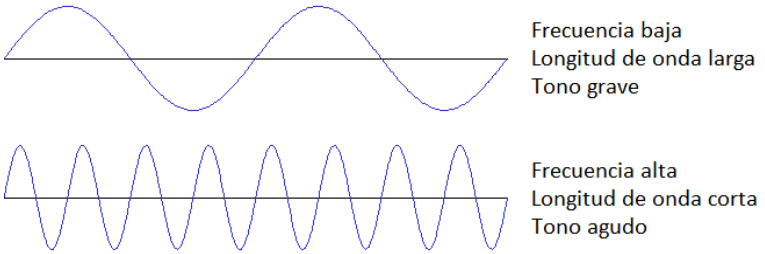


Figura 20: Comparación entre dos ondas de diferente frecuencia

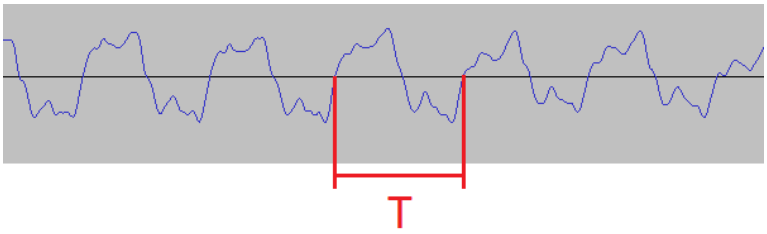


Figura 21: Periodo T de una nota de guitarra

2.2.2.3.1 Notas musicales

En música se utilizan notas musicales que representan frecuencias discretas y previamente establecidas. El grupo básico de notas musicales tienen los nombres de Do, Re, Mi, Fa, Sol, La, y Si, que corresponden a las teclas blancas en un piano (figura 22), y en una pauta musical se identifican por la posición vertical de las figuras musicales, como se muestra en la figura 23. A este conjunto de notas musicales se le conoce como la escala de Do mayor. En el sistema inglés de notación musical, es más común conocer estas mismas notas con los nombres C, D, E, F, G, A y B, respectivamente.

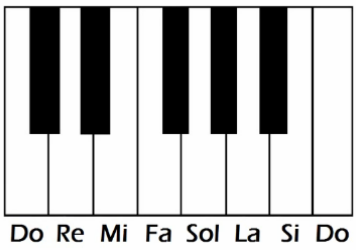


Figura 22: Localización de las notas musicales básicas en el teclado de un piano

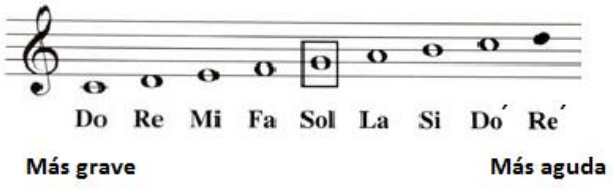


Figura 23: Notas musicales visualizadas en una pauta musical

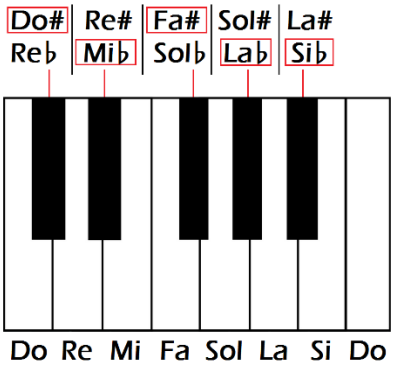


Figura 24: Nombres de las notas localizadas en las teclas blancas y negras de un piano

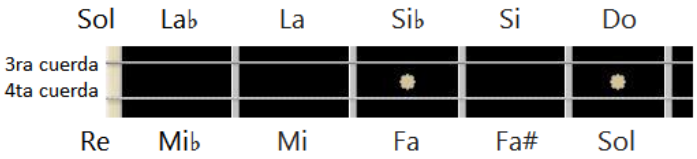


Figura 25: Localización de notas musicales en los primeros 5 trastes de la 3ra y 4ta cuerda de una guitarra

Las teclas negras, que se encuentran entre algunas de las teclas blancas, adquieren su nombre en función de la tecla blanca antes o después de ellas, adjuntando "sostenido" (símbolo: #) o "bemol" (símbolo: ♭) al nombre de la nota. Se usa "sostenido" si es la tecla negra siguiente, y "bemol" si es la tecla negra anterior. Por ejemplo, a la primer tecla negra en la figura 22 se le puede llamar "Do sostenido", escrito Do#, por ser la tecla negra después de Do, o "Re bemol", escrito Re♭, por ser la tecla negra antes de Re.

En la figura 24 se observa que, por lo tanto, las teclas negras técnicamente cuentan con dos nombres. Sin embargo, los músicos le dan preferencia a uno de cada par, los cuales están señalados en la misma figura con recuadros rojos. Estas mismas notas se encontrarán en otros instrumentos bajo un acomodo diferente, mostrando como ejemplo las notas musicales en una guitarra en la figura 25.

Una octava es el grupo de notas de Do a Si, que entonces estaría formada por las notas: Do, Do#, Re, Mi♭, Mi, Fa, Fa#, Sol, La♭, La, Si♭ y Si. A su vez, a este conjunto de notas se le conoce como la escala cromática. El intervalo entre cualquiera de estas notas y la siguiente o la anterior se le denomina "semitono", y un intervalo de dos semitonos es equivalente a un "tono completo".

Las notas de la escala cromática se repiten varias veces en el espectro de notas utilizado en música, donde cada octava está enumerada, como se observa en la figura 26. Por ejemplo, la nota La ubicada específicamente en la 4ta octava se denominaría La4 (figura 27).

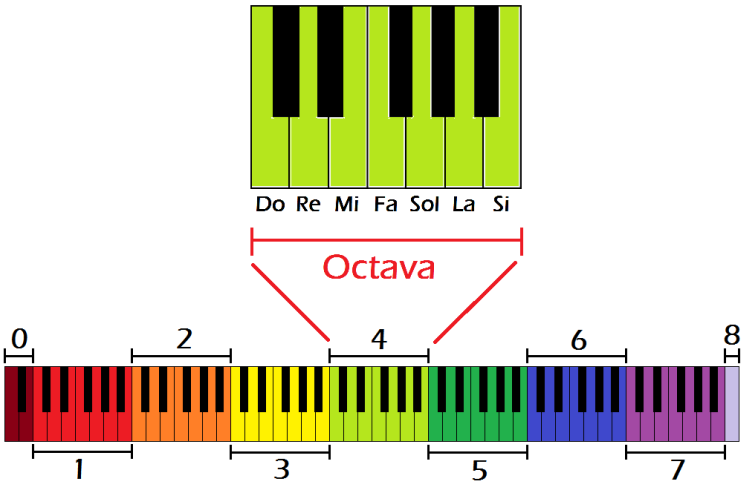


Figura 26: Visualización de la repetición de las notas musicales en un piano de 88 teclas, con cada octava enumerada (https://www.youtube.com/watch?v=WyncfQj8UJc)

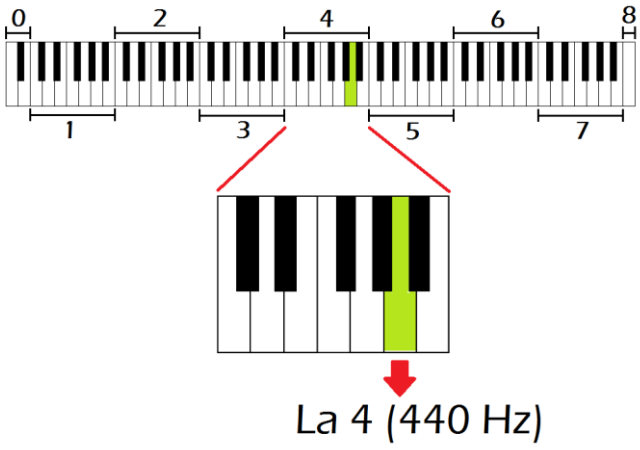


Figura 27: Localización de la nota La4 en un piano de 88 teclas (https://www.youtube.com/watch?v=WyncfQj8UJc)

Una propiedad musical importante es que cualquier nota musical tiene exactamente el doble de frecuencia que la misma nota musical encontrada en la octava anterior. Es decir, si la nota La de la 4ta octava, o La4, tiene una frecuencia de 440 Hz, entonces la nota La5 tiene una frecuencia de 880 Hz, y La3 tiene una frecuencia de 220 Hz.

En afinación con temperamento igual, las frecuencias de las notas en la escala cromática están separadas por intervalos exponenciales iguales usando como factor la raíz doceava de 12. Esto significa que al multiplicar la frecuencia de cualquier nota por $\sqrt[12]{2} = 1.059463094$ ..., encontramos la frecuencia

del siguiente semitono. Al igual que en el caso de la intensidad, la frecuencia es percibida por el oído humano de forma logarítmica, por lo que al utilizar incrementos exponenciales se compensa la percepción logarítmica del oído humano y se perciben incrementos lineales de altura.

La razón por la que se utiliza específicamente el factor de $\sqrt[12]{2}$ es para dividir cada octava en 12 incrementos exponenciales iguales (determinado por el grado de la raíz, en este caso, la raíz doceava), representando los 12 semitonos contenidos en una octava, y para que cada incremento de una octava completa resulte en el doble de frecuencia (determinado por el "2" dentro de la raíz).

La frecuencia de una nota musical, en función del número de tecla donde se ubica en un piano de 88 teclas, se puede calcular mediante la siguiente fórmula:

$$f(n) = 440\ Hz\ x\left(\sqrt[12]{2}\right)^{n-49} \tag{19}$$

donde $f(n)$ es la frecuencia buscada y n es el número de tecla, siendo La0 la tecla 1. n incrementa en intervalos de semitono, por lo que la tecla 2 sería Si♭0.

Esta fórmula se obtiene a partir de que el sistema de afinación de acuerdo al estándar ISO 16:1975 utiliza temperamento igual y está basado en La4 = 440 Hz, que es la tecla número 49 en un piano. Es decir que las frecuencias de todas las notas musicales se obtienen respecto a La4, la cual tiene asignada una frecuencia de 440 Hz.

Los intervalos de semitono se pueden dividir en intervalos más pequeños llamados "cents". Estos son los incrementos mínimos considerados en música, y equivalen a una centésima de semitono, que viene siendo un incremento en la frecuencia por un factor de $\sqrt[100]{\sqrt[12]{2}}$, o sea, $\sqrt[1200]{2}$. Los cents se utilizan en casos como la comparación de las alturas de notas musicales en diferentes sistemas de afinación.

2.2.2.4 Timbre

El timbre es el sonido característico de un instrumento musical o una voz. En otras palabras, es la propiedad con la cual distinguimos entre dos instrumentos o dos voces aunque estén reproduciendo exactamente la misma nota musical.

En el dominio del tiempo, sonidos con diferente timbre presentarán una diferente forma de onda. En la figura 28 se observan algunos ejemplos de formas de onda de diferentes instrumentos musicales sintetizados por software (instrumentos virtuales incluidos en el software "Mixcraft" de Acoustica®).

Sin embargo, en la música es más útil el análisis desde la perspectiva del espectro de frecuencias, que se obtiene a través de la transformada de Fourier, o la transformada rápida en el caso específico de llevar a cabo el cálculo con software, explicados en las secciones 2.1.2.2 y 2.1.2.2.1.

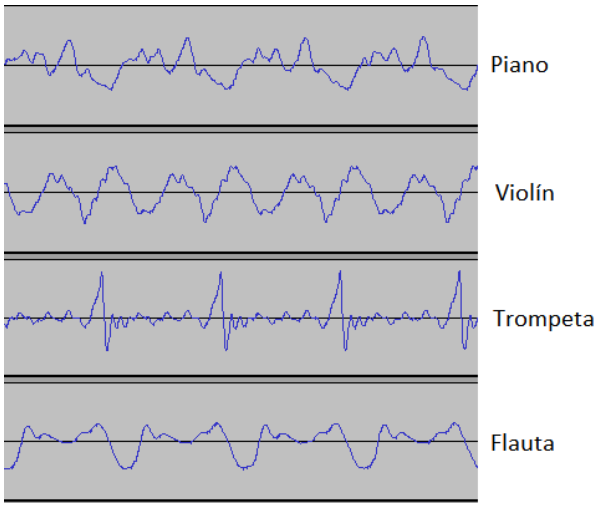


Figura 28: Formas de onda de diferentes instrumentos musicales (sintetizados) tocando la misma nota musical (Do4)

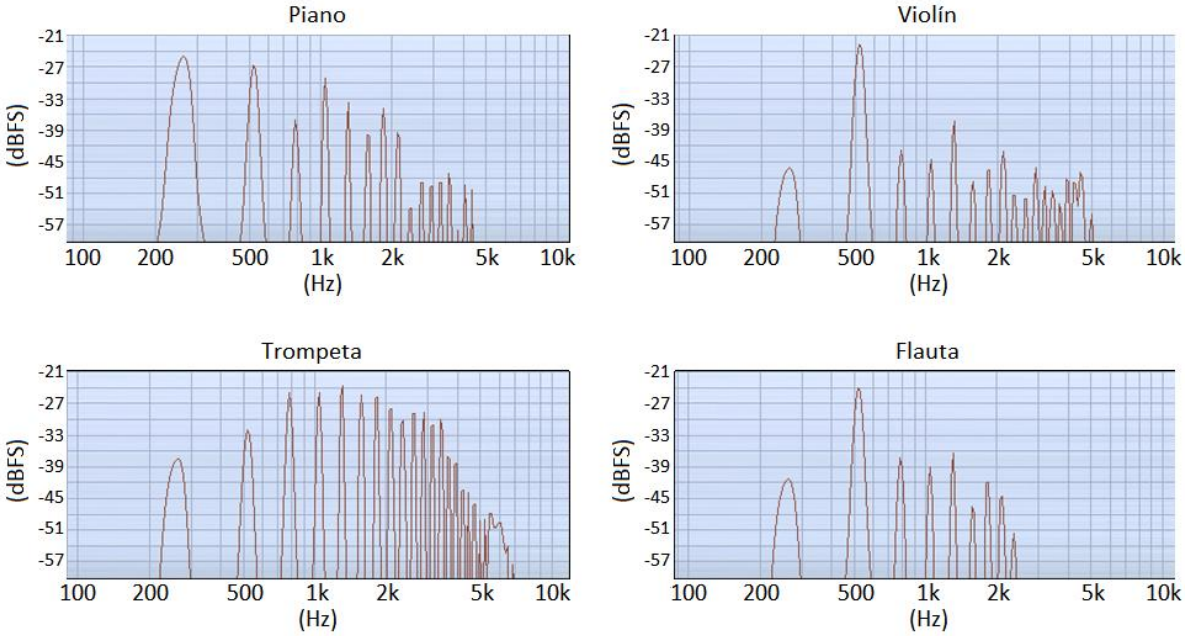


Figura 29: Espectros de frecuencia de los casos mostrados en la figura 28

Anteriormente se mencionó que una onda senoidal tiene una sola componente de frecuencia, y es por esta razón que en música se le llama “tono puro” al sonido producido por una onda senoidal.

En la figura 29 se muestran los espectros de frecuencia de los mismos casos de la figura 28, obtenidos a través del software “Voxengo Spectrum Analyzer”. Esta figura muestra un aspecto diferente a la figura 7 (pág. 8), a pesar de ambos tratarse de espectros de frecuencia.

Esto se debe a que el espectro en la figura 7 corresponde a una señal continua y un caso matemático ideal, mientras que en la práctica se obtiene el espectro a partir de una señal discreta, y se cuenta con una precisión limitada debido a la discretización. Adicionalmente, la figura 7 muestra la frecuencia y la amplitud en escala lineal, mientras que la figura 29 muestra el eje de frecuencia en escala logarítmica, y la amplitud en decibeles.

En el dominio de la frecuencia, el timbre de un sonido dependerá del conjunto de frecuencias que forman su espectro y sus respectivas amplitudes. Por lo tanto, el timbre de un sonido, su forma de onda, y su espectro de frecuencia, están todos relacionados entre sí.

Los sonidos que se perciben con una altura específica están compuestos principalmente de componentes frecuenciales armónicas, es decir, cuyas frecuencias son múltiplos enteros de la frecuencia de la componente fundamental, que es la componente de más baja frecuencia en el espectro de frecuencias. Al mismo tiempo, la fundamental determina la frecuencia de la onda en el tiempo, y por lo tanto su altura.

Por ejemplo, en todos los casos de la figura 29 se está reproduciendo la nota Do4, cuya frecuencia es 261.626 Hz, y esta misma frecuencia es la componente más grave que se observa en el espectro. Los armónicos se encuentran en 523.251 Hz (261.626 x 2), 784.877 Hz (261.626 x 3), etc.

Sin embargo, las diferentes amplitudes de sus componentes es lo que nos permite reconocer entre el sonido de los diferentes instrumentos. Una trompeta produce un timbre más agudo que un piano tocando la misma nota musical, pues la trompeta tiene mayor potencia en sus armónicos altos, particularmente observable en el rango entre 2 kHz y 5 kHz en la figura 29.

Los sonidos ruidosos, como los de las percusiones no tonales, cuentan con mucho contenido inarmónico, es decir, en su espectro no se identifica una fundamental a la cual sean proporcionales las componentes frecuenciales que componen el sonido. Ejemplos de percusiones no tonales serían la tarola y el platillo (figura 30), a diferencia de las percusiones tonales como el xilófono.

Hay que considerar que las componentes frecuenciales del tono generado por un instrumento musical varían con el tiempo, por lo que visualizar un tono o una obra musical requiere que se esté calculando el espectro de frecuencia repetidamente para pequeñas porciones de audio.

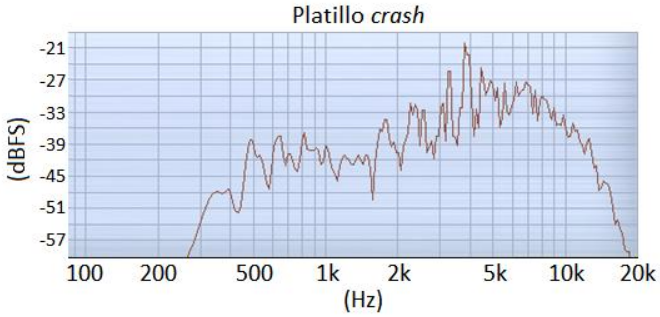


Figura 30: Espectro de frecuencia de un platillo crash

2.2.3 Percepción del sonido por el oído humano

El sonido es percibido por un individuo después de pasar por las tres secciones del oído humano: el oído externo, el oído medio, y el oído interno (figura 31).

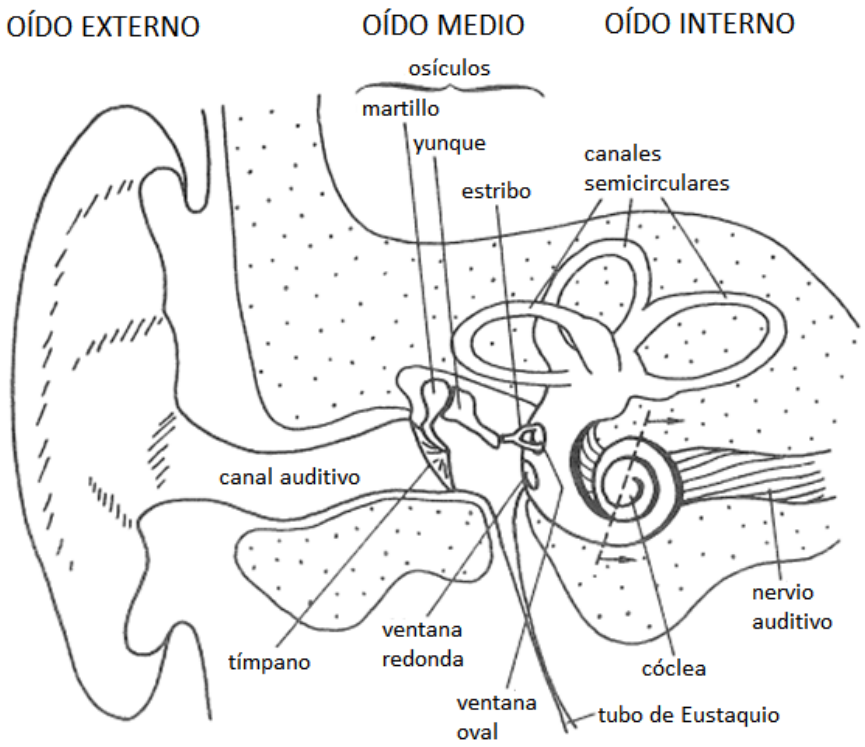


Figura 31: Partes del oído externo, medio, e interno (Psychoacoustics, Facts and models, Zwicker, p.24)

El oído externo está constituido por la oreja y el canal auditivo, y termina en el tímpano. Las ondas de presión entran por el canal auditivo, mientras la oreja contribuye transfiriendo ondas mecánicamente al canal auditivo, hasta que las ondas impactan el tímpano, causando que éste vibre acorde a la forma de la onda sonora incidente.

En el oído medio se encuentran un grupo de huesecillos: el martillo, el yunque, y el estribo, a los cuales el tímpano transfiere su vibración. Estos huesecillos tienen la función de amplificar las vibraciones, lo cual es necesario pues las vibraciones serán transferidas a un medio más espeso en el oído interno a través de la ventana oval.

El oído interno es la sección más compleja, y el órgano clave es la cóclea, que tiene una forma de espiral. Dentro de ella se encuentra enrollada la membrana basilar, que se observa en la figura 32 como si se hubiera desenrollado.

La membrana basilar tiene una variación gradual de anchura y rigidez, siendo más estrecha y rígida en su base, y más ancha y flexible en su ápice. Esto causa que diferentes frecuencias resuenen en diferentes puntos de ella. Específicamente, las frecuencias altas resuenan en su base, y las frecuencias bajas resuenan en su ápice.

La vibración hacia arriba y hacia abajo de la membrana basilar en diferentes puntos de la misma hace vibrar al órgano de Corti (figura 33) en la misma dirección, y al mismo tiempo hace vibrar a la membrana tectorial hacia los lados. El movimiento combinado del órgano de Corti y la membrana tectorial provoca un movimiento lateral en los cilios de las células ciliadas, que envían señales eléctricas a través de las fibras del nervio auditivo. El sonido es percibido una vez que estas señales eléctricas llegan al cerebro.

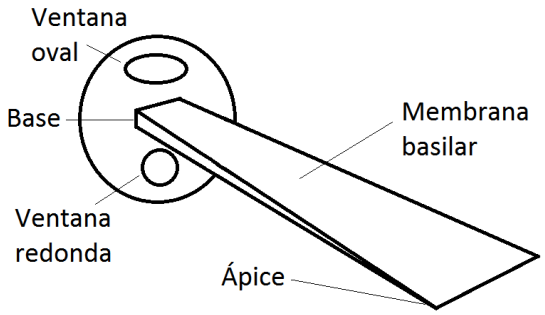


Figura 32: Visualización de la membrana basilar desenrollada

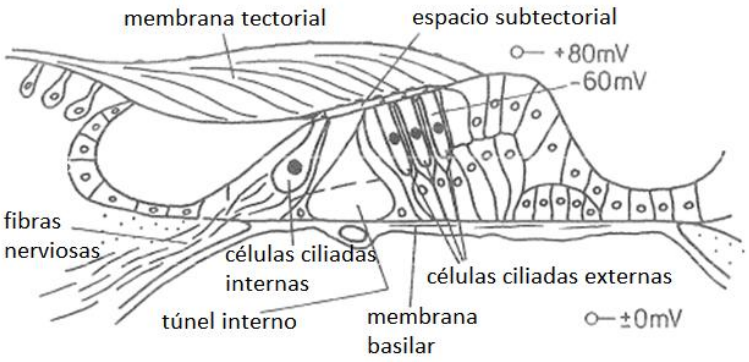


Figura 33: Diagrama del órgano de Corti y los tejidos que lo rodean (Psychoacoustics, Facts and models, Zwicker, p.26)

El papel que lleva a cabo el oído interno a través de la membrana basilar, el órgano de Corti, la membrana tectorial, y las células ciliadas, es equivalente a realizar análisis espectral, pues la membrana basilar resuena en diferentes puntos dependiendo de las frecuencias que la hacen vibrar, y las células ciliadas que se estimulen dependerá del punto (o los puntos) en el que esté resonando la membrana basilar. Es debido a este funcionamiento que somos capaces, por ejemplo, de distinguir entre varios instrumentos musicales tocando simultáneamente aunque la información llegue a nuestro tímpano combinada en una sola forma de onda.

Como se mencionó anteriormente, la percepción de frecuencia del oído humano es logarítmica, por lo que se requieren incrementos exponenciales de frecuencia para percibir incrementos lineales de altura.

El oído humano generalmente es capaz de percibir frecuencias entre 20 Hz y 20 kHz, aunque con sensibilidad reducida en los límites inferior y superior. Este rango se reduce conforme aumenta la edad.

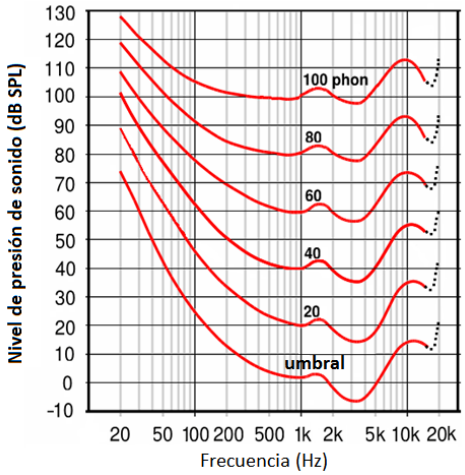


Figura 34: Curvas de igual volumen, ISO 226:2003

En la figura 34 se muestran las curvas de igual volumen de audibilidad humana de acuerdo al estándar ISO 226:2003. Estas curvas se pueden considerar lo opuesto a una respuesta en frecuencia: en vez de medir la respuesta de tonos puros con la misma amplitud, esta gráfica describe los niveles de presión sonora requeridos a diferentes frecuencias para que el individuo perciba los tonos con el mismo volumen.

Por ejemplo, se puede interpretar de los valores elevados a frecuencias bajas que se requiere que estos tonos se reproduzcan con un nivel muy alto para percibirlos igual que otras frecuencias en el rango medio a menor nivel de presión sonora, lo que significa una audibilidad reducida en las frecuencias bajas.

2.2.3.1 Diferencia apenas perceptible de frecuencia

La diferencia apenas perceptible (*JND, "just noticeable difference")* por el ser humano entre dos frecuencias está en función de las frecuencias como tal [9], como se muestra en la figura 35. En el eje horizontal se muestra la frecuencia base f_1, y en el eje vertical se muestra la diferencia de frecuencia apenas perceptible. Las líneas punteadas muestran las diferencias en porcentaje, que son más relevantes para el oído humano por su percepción logarítmica de frecuencia.

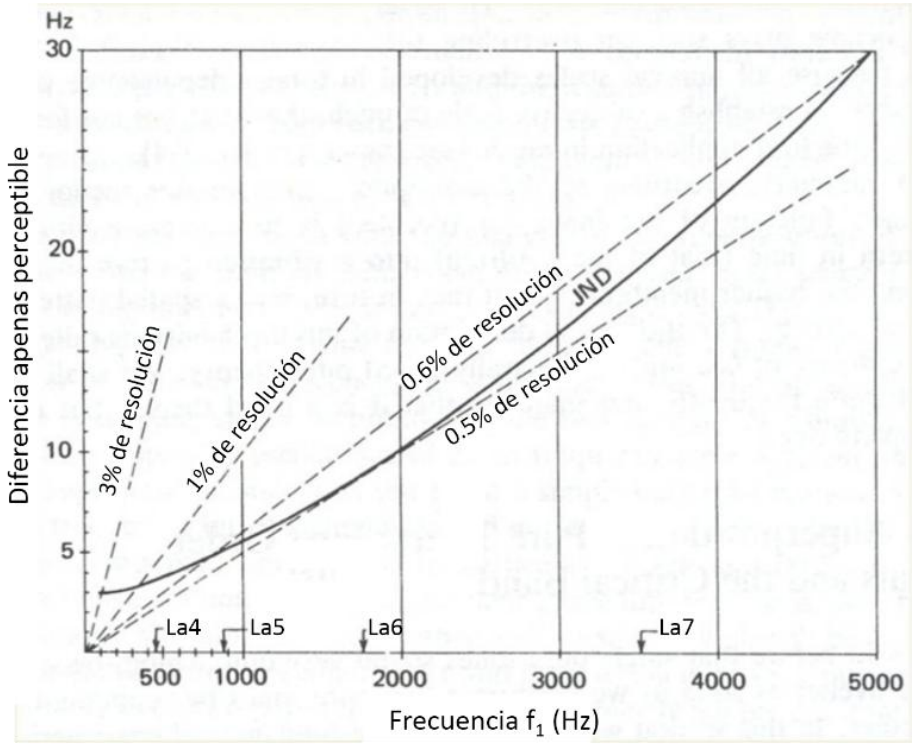


Figura 35: Curva de diferencia apenas perceptible de frecuencia (The Physics and Pyschophysics of Music – An introduction, J. G. Roederer, p. 36)

Observamos que la mínima diferencia perceptible de frecuencia está en el extremo izquierdo y es de 3 Hz, mientras que la mínima diferencia *relativa* de frecuencia perceptible se encuentra alrededor de la nota La6 (1760 Hz), donde se puede distinguir un 0.5% de diferencia de frecuencia.

Recordando que los cents (unidad descrita en la sección 2.2.2.3.1) equivalen a un incremento de frecuencia por un factor de $\sqrt[1200]{2}$ cents, una relación en porcentaje se puede convertir a cents a través de la siguiente ecuación:

$$n_{cents} = log_{\sqrt[1200]{2}}\left(1 + \frac{k_{\%}}{100\%}\right) \tag{20}$$

que se puede simplificar a:

$$n_{cents} = 1200 \cdot log_2\left(1 + \frac{k_{\%}}{100\%}\right) \tag{21}$$

donde n_{cents} es la cantidad en cents y $k_{\%}$ es la proporción en porcentaje.

Utilizando la ecuación 21:

$$n_{cents} = 1200 \cdot log_2\left(1 + \frac{0.5\%}{100\%}\right) = 1200 \cdot log_2(1.005) = 8.63\ cents$$

Es decir, la diferencia relativa mínima de frecuencia perceptible por el ser humano, que es de 0.5%, equivale a 8.63 cents.

Considerando que se tiene planeado sintetizar eléctricamente tonos musicales, será necesario estudiar aquellos conceptos y dispositivos que unen el dominio eléctrico con el acústico, que será el tema de la siguiente sección.

2.3 Electroacústica

Entre los tantos dispositivos cuya función es convertir de un tipo de energía a otra, existen transductores capaces de convertir energía acústica a eléctrica, donde la variación de presión respecto a la presión atmosférica se puede traducir a magnitudes positivas y negativas de voltaje o corriente. Estas magnitudes guardan relación con la onda sonora original, y la información en ambos dominios se puede estudiar a través de análisis espectral.

Esto abre las puertas a utilizar la electrónica para transferir, analizar, almacenar y modificar las señales acústicas, a través de las herramientas de señales y sistemas, y procesamiento digital de señales, todo dentro del dominio de la electrónica, para finalmente regresar del dominio eléctrico al acústico.

Alternativamente, es posible generar señales eléctricas directamente con un circuito eléctrico o desde un dispositivo digital programable (con su debida conversión a señal analógica), y al igual que cualquier señal eléctrica, esta se puede enviar directamente a altavoz para producir sonido.

2.3.1 Señal analógica

Se considera que una señal (refiriéndose usualmente, aunque no estrictamente, al dominio eléctrico) es analógica cuando está en función del tiempo.

Se debe tener en consideración que el tiempo es un parámetro continuo, es decir, no está formado por elementos fundamentales discretos. Las señales analógicas, por estar en función del tiempo, adquieren esta propiedad, y describen la evolución temporal de diferentes fenómenos en el mundo físico.

2.3.2 Respuesta en frecuencia

Las características de un sistema físico ocasionarán que este tenga una cierta respuesta a diferentes frecuencias de oscilaciones mecánicas, eléctricas, electromagnéticas, etc., dependiendo del caso.

Para caracterizar tal sistema, se somete a una prueba donde se introducen ondas senoidales en un rango específico de frecuencias, todas con la misma amplitud, y se toma lectura de la salida. Con ello, se genera una curva que representa la forma en la que el sistema naturalmente amplifica o atenúa diferentes frecuencias, o la sensibilidad a diferentes frecuencias en caso de ser un transductor.

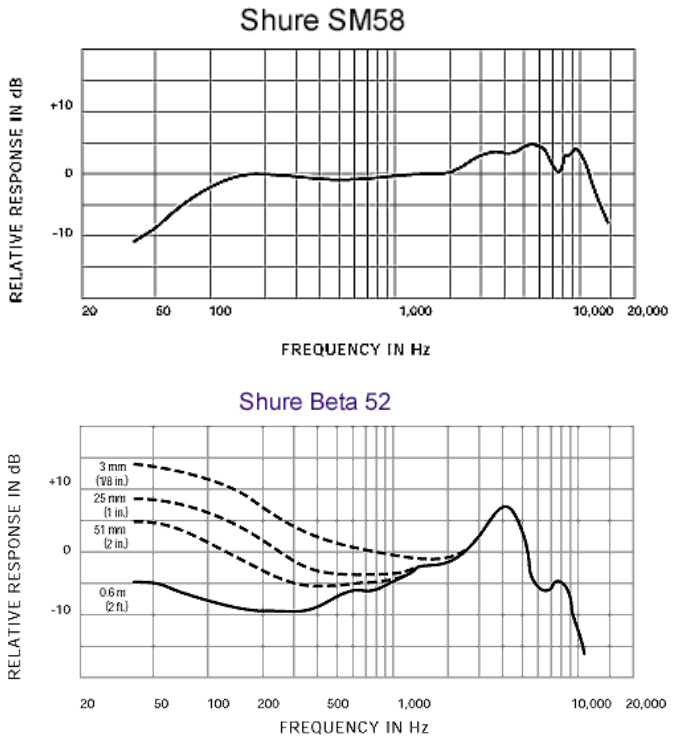


Figura 36: Respuestas de un micrófono Shure® SM58 para voz y un micrófono Shure® Beta 52 para bombo de batería (http://www.shureasia.com/en/products/microphones/sm58-vocal-microphone, http://www.shure.co.uk/products/microphones/beta_52a)

En electroacústica, ejemplos comunes de respuesta en frecuencia son las magnitudes con las que un micrófono o un altavoz pueden captar o reproducir, respectivamente, diferentes frecuencias sonoras. En la figura 36 se comparan las respuestas en frecuencia de un micrófono para voz y un micrófono para bombo de batería, apreciando en el segundo caso una mayor respuesta hacia frecuencias bajas.

Se observa que los decibeles empleados en el eje vertical son relativos a la respuesta a 1 kHz, es decir, representan la cantidad de decibeles por arriba o por debajo de la medición que se tomó a 1 kHz, cualquiera que haya sido. Por esta misma razón se muestra una respuesta de 0 dB a 1 kHz, recordando que 0 dB se interpreta como un nivel igual a la referencia utilizada.

Con esta información, se puede conocer de antemano la respuesta que tendría un dispositivo hacia cierto rango de frecuencias, por lo que se puede decidir si el dispositivo es apto para la aplicación para la cual se está contemplando.

2.3.3 Transductores

Los transductores electroacústicos, es decir, que transforman energía sonora a eléctrica o viceversa, se pueden clasificar de acuerdo a diversos parámetros, tales como el componente transductor, la respuesta en frecuencia, y la direccionalidad. Por consecuencia, existe una gran variedad de micrófonos y altavoces.

En esta sección se describirán los fundamentos básicos de operación específicamente de los micrófonos y altavoces cuyo elemento transductor es el inductor.

2.3.3.1 Micrófono

En la figura 37 se observa la estructura de un micrófono de inductor, conocido comúnmente como "micrófono dinámico". Las ondas sonoras impactan sobre un diafragma que vibra acorde a las variaciones de presión. Un inductor está acoplado al diafragma, por lo que vibra junto a él. Por el centro del inductor pasa un imán, y el movimiento del inductor ocasiona que este perciba el campo magnético del imán como un campo magnético variante, generando una fuerza electromotriz por efecto de inducción electromagnética, descrita por la ecuación de Maxwell-Faraday:

$$\varepsilon = -N\frac{d\Phi_B}{dt}, \tag{22}$$

donde ε es la fuerza electromotriz resultante, N es el número de vueltas del inductor, y $d\Phi_B$ es la variación del campo magnético.

Se toma la lectura de la salida en los extremos del inductor, y la forma de onda de la corriente generada será similar (más no una reproducción perfecta) de la forma de onda de la señal sonora.

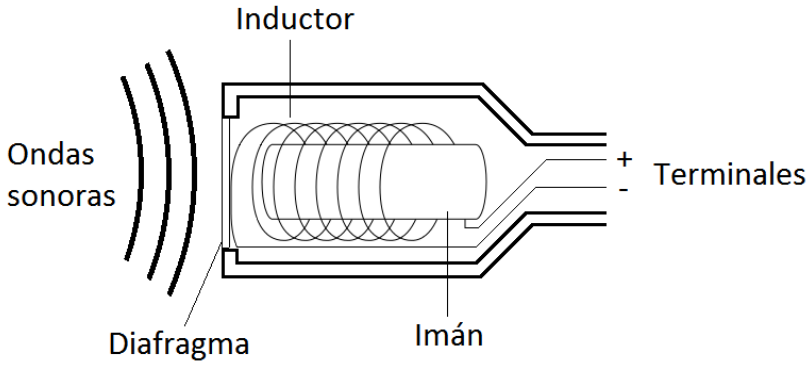


Figura 37: Estructura de un micrófono dinámico

Típicamente, se conecta un micrófono a un altavoz para una amplificar el sonido original producido por una fuente sonora, como un cantante o un instrumento musical, o a un sistema digital para almacenar la señal acústica, por lo que es la herramienta elemental para la grabación de sonido. Sin embargo, es indispensable siempre tener en cuenta su respuesta en frecuencia para asegurar que se está captando el rango de frecuencias requerido.

2.3.3.2 Altavoz

Un altavoz es un transductor que convierte una señal eléctrica a una señal sonora. Está compuesta principalmente por un inductor, un imán, un cono, y un soporte mecánico llamado ‘‘araña’’.

La señal de entrada pasará por un inductor, el cual va generar un campo magnético cuya intensidad dependerá de la corriente, de acuerdo a la Ley de Ampere:

$$\oint_C \mathbf{B} \cdot \mathrm{d}\boldsymbol{\ell} = \mu_o \int\int_S \mathbf{J} \cdot \mathrm{d}\mathbf{S} = \mu_o I_{enc}\,, \tag{23}$$

donde **B** es el campo magnético, d$\boldsymbol{\ell}$ es el diferencial de longitud, μ_o es la constante magnética, **J** es la densidad total de corriente, d**S** es el diferencial de superficie, e I_{enc} es la corriente total pasando por la superficie S.

En la figura 38 se muestra la estructura de un altavoz. La variación del campo magnético del inductor, causada por la señal eléctrica variante a la entrada, causará que el inductor constantemente sea atraído y rechazado por el imán, produciendo un movimiento acorde a la variación de la señal de entrada. El cono se moverá junto con el inductor, empujando y jalando el aire junto a él, lo que produce las ondas de variación de presión que constituyen una onda sonora. La araña es un soporte mecánico que mantiene suspendidos el cono y el inductor, permitiendo cierta libertad de movimiento para la vibración.

Por su construcción, es posible conectar y usar un altavoz como micrófono, y un micrófono dinámico como altavoz, pues los componentes básicamente son los mismos. Sin embargo, no darán un buen rendimiento pues no estarían siendo utilizados para la aplicación para la que fueron diseñados.

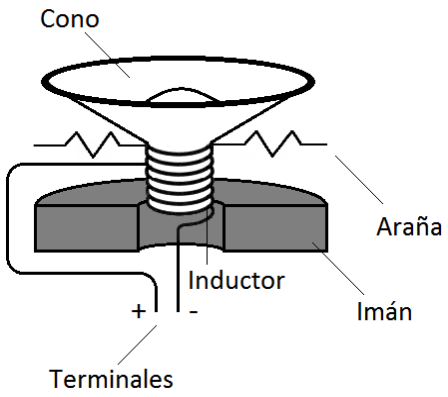


Figura 38: Estructura de un altavoz

La respuesta en frecuencia de un altavoz depende principalmente de la construcción del altavoz (materiales, tamaño, rigidez de la araña, etc.) y no tanto de su transductor. Por ejemplo, un altavoz para bajos (frecuencias bajas) se diseña para enfatizar las componentes de frecuencia baja y reducir o eliminar las componentes de frecuencia alta de la señal de entrada, esto para asegurar una respuesta buena en el rango para el que fue diseñado. Por lo tanto, si se introduce a tal altavoz una señal con componentes de alta frecuencia, la bocina será incapaz de reproducirlas fielmente, produciendo un timbre diferente al originalmente esperado.

Los altavoces se pueden clasificar por su respuesta en frecuencia en 4 rangos principales: *Subwoofer* (típicamente 20-200 Hz), *Woofer* (típicamente 40-1000 Hz), *Midrange* (típicamente 300-5000 Hz), y *Tweeter* (típicamente 2 kHz a 20 kHz).

2.3.4 Filtros analógicos pasa-bajas y pasa-altas

Un filtro es un circuito con una entrada y una salida, cuyo propósito es modificar las componentes frecuenciales de la señal de entrada.

Es posible construir un filtro utilizando únicamente un resistor y un capacitor, o un resistor y un inductor, en cuyo caso sería un *filtro pasivo*. Sin embargo, al conectar este circuito a otro dispositivo no habrá un acoplamiento adecuado de impedancias, por lo que cambiarán sus propiedades. Es por ello que puede resultar más conveniente utilizar un *filtro activo* (figuras 39 y 40), que por la utilización de un amplificador operacional poseen una impedancia de salida muy baja. Hay que tomar en cuenta que los circuitos mostrados están en configuración inversora, por lo que el voltaje a la salida tendrá el signo opuesto.

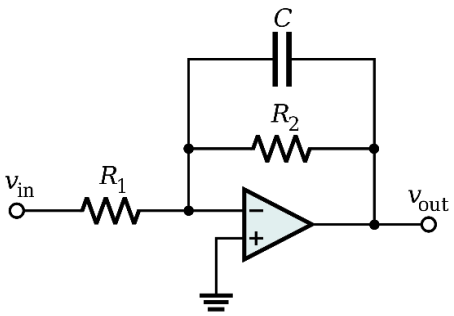


Figura 39: Filtro activo inversor pasa-bajas

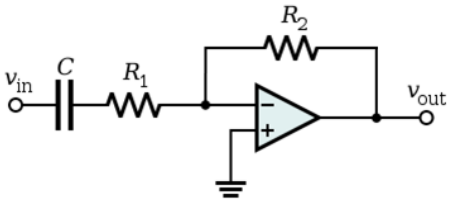


Figura 40: Filtro activo inversor pasa-altas

Cada filtro poseerá su respuesta en frecuencia característica, con los decibeles del eje vertical referenciados a la señal original, por lo que una respuesta de 0 dB en una frecuencia específica significaría que esa frecuencia no será atenuada ni amplificada.

En un filtro se identifican dos parámetros principales: la ganancia y la frecuencia de corte. La ganancia es una amplificación que afecta equitativamente a todas las frecuencias, mientras que la frecuencia de corte es la frecuencia en la que se tendrá una atenuación de -3 dB. Un filtro pasa-bajas principalmente atenuará las frecuencias mayores a la frecuencia de corte, mientras que un filtro pasa-altas principalmente atenuará las frecuencias menores a la frecuencia de corte.

La conexión en serie de un filtro pasa-bajas y un filtro pasa-altas, donde la frecuencia de corte del filtro pasa-bajas es mayor a la del filtro pasa-altas, formará un filtro pasa-banda.

2.3.4.1 Diseño de filtros por diagrama de Bode

El diagrama de Bode es una aproximación a la respuesta en frecuencia de un filtro, el cual es muy sencillo de realizar. Mientras que existen diagramas de Bode para magnitud y para fase, en esta sección se explicará específicamente el diagrama de Bode para magnitud. Sin embargo, hay que mantener en cuenta que el filtro producirá cambios de fase en las componentes espectrales.

Primero, se necesita decidir la ganancia y la frecuencia de corte deseadas para el filtro. Basado en las figuras 39 y 40, la ganancia G para ambos tipos de filtro está dada por:

$$G = -\frac{R_2}{R_1} \tag{24}$$

El signo negativo es debido a la configuración inversora.

Para el diseño del filtro por diagrama de Bode, es recomendable obtener la ganancia en decibeles dada por:

$$G_{dB} = 20\, log_{10}\, |G| \tag{25}$$

Para el filtro pasa-bajas de la figura 39, la frecuencia de corte f_c (no confundir con la frecuencia de portadora f_c) estaría dada por:

$$f_c = \frac{1}{2\pi R_2 C} \tag{26}$$

Mientras que para el filtro pasa-altas de la figura 40 se obtendría mediante:

$$f_c = \frac{1}{2\pi R_1 C} \tag{27}$$

Se observa que cada ecuación involucra dos componentes electrónicos, de los cuales uno se puede elegir libremente, y el otro se obtiene por despeje.

El diagrama de Bode se realiza utilizando un eje logarítmico de frecuencias. El primer paso es localizar la frecuencia de corte, y establecer un punto cuyas coordenadas son esta frecuencia y la ganancia del filtro en decibeles. En el caso de un filtro pasa-bajas, se traza una línea recta de este punto hacia la izquierda, y desde ese mismo punto una línea diagonal hacia la derecha con una pendiente de -20 dB por década, donde una década es una frecuencia 10 veces mayor. En el caso de un filtro pasa-altas se traza una línea recta de este punto hacia la derecha, y desde ese mismo punto una línea diagonal hacia la izquierda con una pendiente de 20 dB por década.

La respuesta en frecuencia real del circuito será una aproximación a estos trazos, suavizando la respuesta justo debajo del punto localizado en la frecuencia de corte, para obtener una respuesta 3 dB menor.

Por ejemplo, para un filtro pasa-bajas con una ganancia de 1, es decir, 0 dB, y una frecuencia de corte de 1 kHz, la aproximación de su respuesta en magnitud por diagrama de Bode, y su verdadera respuesta en frecuencia, se observan en la figura 41, en azul y rojo respectivamente.

Para el cálculo de los componentes, se decide utilizar un resistor R_2 de 4.7 kΩ, por lo que para una ganancia (negativa) de 1, R_1 debe tener este mismo valor.

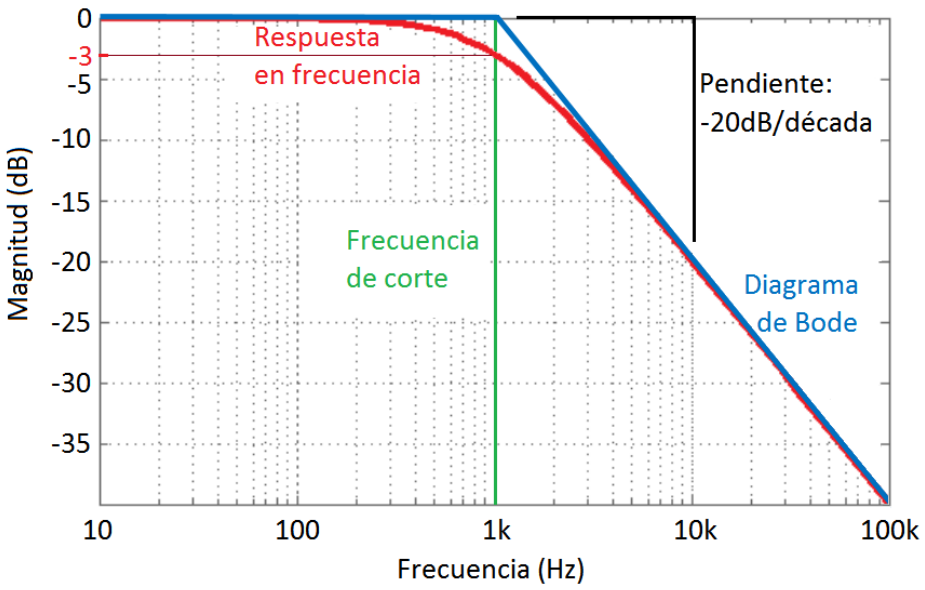


Figura 41: Diagrama de Bode para magnitud y respuesta en frecuencia de un filtro pasa-bajas con f_c = 1 kHz y ganancia de 0 dB

Posteriormente, se despeja C de la ecuación 26 y se resuelve:

$$C = \frac{1}{2\pi f_c R_2} = \frac{1}{2\pi(1\ kHz)(4.7\ k\Omega)} = 33.862x10^{-9}F \approx 33nF$$

Un filtro en el cual la recta inclinada de su diagrama de Bode tiene una pendiente de ±20 dB por década (+20 para filtro pasa-altas, -20 para filtro pasa-bajas) se denomina un filtro de primer orden. Sin embargo, conectar dos filtros iguales en serie aumenta la pendiente de la recta a ±40 dB resultando en un corte más brusco de frecuencias después o antes de su frecuencia de corte, y se denominaría un filtro de segundo orden. Cada filtro adicional conectado en serie aumenta su orden por 1.

Alternativamente, existen configuraciones donde se puede obtener un filtro de 2do orden con un solo amplificador operacional, como la topología Sallen-Key (figura 42).

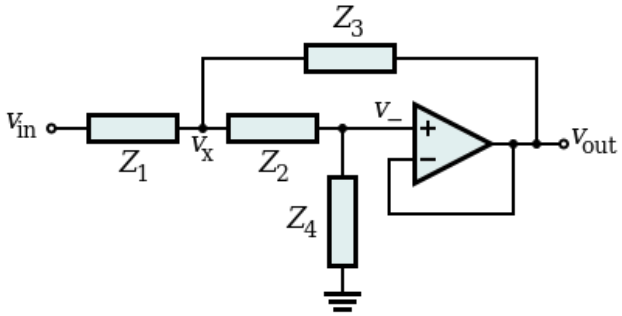


Figura 42: Topología Sallen-Key para filtros electrónicos

2.3.5 Osciladores

Un oscilador es un dispositivo electrónico capaz de producir una señal eléctrica variante periódica. En la figura 43 se muestra un ejemplo.

Los sintetizadores comúnmente cuentan con osciladores electrónicos (analógicos o digitales) capaces de generar formas de onda sencillas, usualmente de onda senoidal, onda cuadrada y onda triangular.

Estos osciladores pueden estar conectados directamente a la salida para producir sonido, o pueden estar conectados como entrada a otro oscilador para llevar a cabo una modulación en amplitud o en frecuencia, produciendo diversos efectos musicales.

Para el caso de efectos musicales generalmente se utilizan osciladores de baja frecuencia (LFO, "*low frequency oscillator*"), cuya frecuencia es menor a 20 Hz (por debajo del rango audible).

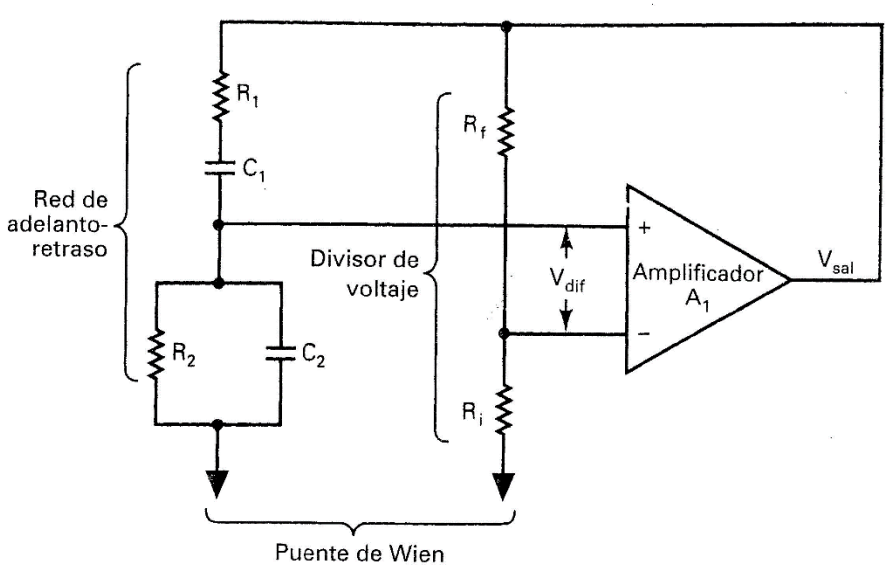


Figura 43: Oscilador de puente de Wien (W. Tomasi y V. García Bisogmo, Sistemas de comunicaciones, pág. 55)

2.4 Electrónica digital

Por la naturaleza digital del sintetizador que se diseñará, es necesario conocer los conceptos que involucran la transferencia de información del dominio analógico al digital y viceversa, para asegurar un procesamiento y resultados adecuados.

Tales conceptos se revisarán en esta sección, y además se presentará el tipo de dispositivo digital con el que se realizará la síntesis de sonido para el caso particular de la presente tesis.

2.4.1 Señal digital

En un sistema digital, la información se almacena de forma discreta, es decir, en unidades llamadas 'bits' que son la unidad mínima de información digital, y cuyo valor puede ser "0" o "1". Cualquier tipo de información manipulada o almacenada por un sistema digital, sean números, palabras, imágenes, video, sonido, etc., es representada por dicho sistema a través de largas cadenas de ceros y unos lógicos, que se pueden analizar como números en sistema binario.

Por lo tanto, una señal analógica captada y llevada a un sistema digital, o una señal sintetizada directamente en el dominio digital, estarán almacenadas en este formato.

2.4.2 Conversión analógico a digital

El almacenamiento y/o el procesamiento de una señal eléctrica en el dominio digital requieren de la previa discretización de la señal analógica, a través de un convertidor analógico-a-digital (ADC).

En un cierto instante de tiempo, el ADC toma lectura del voltaje a la entrada y lo almacena (aun analógicamente) mediante un circuito *sample and hold* (de muestreo y retención) como se muestra en la figura 44.

Posteriormente, se lleva a cabo un proceso iterativo de incrementos de bit en un vector binario, hasta que se determina que el equivalente en voltaje del número binario es suficientemente aproximado al voltaje retenido. Este procedimiento lo lleva a cabo un circuito como el mostrado en la figura 45.

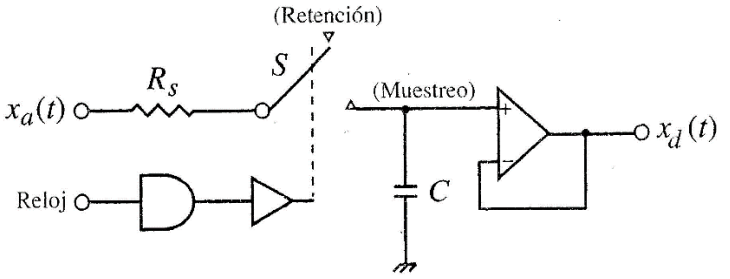


Figura 44: Circuito de muestreo y retención (S. Mitra, Procesamiento de señales digitales, pág. 207)

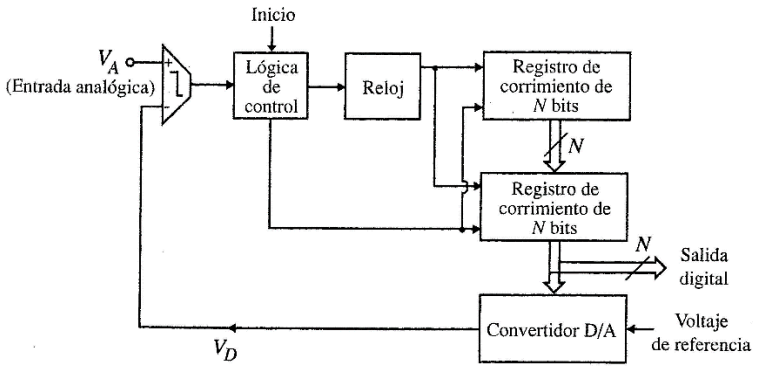


Figura 45: ADC por aproximaciones sucesivas (S. Mitra, Procesamiento de señales digitales, pág. 210)

Es importante mantener en cuenta este procedimiento, pues de él depende la rapidez con la que se podrán tomar lecturas repetidamente. Esto no presentaría problema para una señal constante o de variación lenta, pero puede presentar una limitante al intentar convertir una señal con una variación considerablemente rápida.

Cada lectura individual tomada por el ADC se le denomina "muestra". El tiempo que le toma a un ADC tomar una muestra y convertirla a formato digital dependerá del funcionamiento específico del ADC, y se denomina "periodo de muestreo", con símbolo T_s. Entonces, la tasa de muestreo f_s está dada en Hz por:

$$f_s = \frac{1}{T_s}, \tag{28}$$

y representa el número de muestras que el dispositivo es capaz de tomar en un segundo.

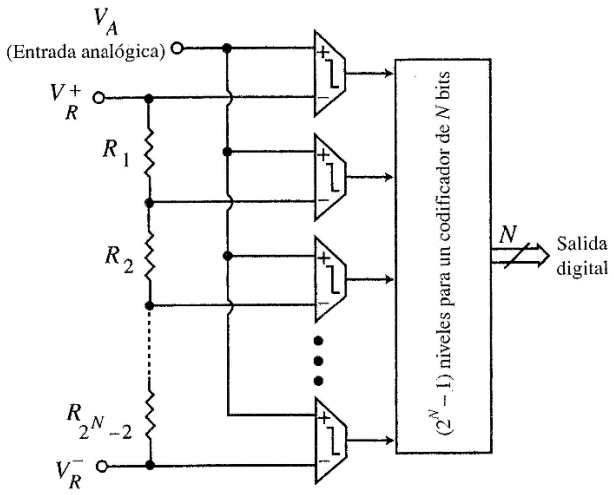


Figura 46: ADC flash (S. Mitra, Procesamiento de señales digitales, pág. 209)

En el caso particular de un ADC *flash* (figura 46), la conversión es inmediata, por lo que la única limitación temporal sería la velocidad de respuesta de los componentes. Sin embargo, este tipo de ADC requiere una gran cantidad de componentes para altas resoluciones, por lo que comúnmente no son una opción viable en aplicaciones de alta integración.

Para la representación de valores discretos de magnitud, el ADC contará con una cantidad específica de bits, denominada "profundidad de bits". De esta característica dependerá la precisión con la que el ADC podrá representar valores de voltaje en un cierto rango.

La resolución es el mínimo incremento de voltaje que un ADC es capaz de producir con un incremento unitario en el vector de bits, y está dada por:

$$Resolución = \frac{V_{ref}}{2^n - 1} \tag{29}$$

donde V_{ref} es el voltaje de referencia del convertidor, y n es la profundidad de bits.

Como ejemplo, en el caso de una profundidad de 3 bits, únicamente existirían 8 (2^3) niveles discretos de voltaje que el sistema digital podría representar, que es el número de combinaciones binarias posibles desde 000 hasta 111.

Con esta profundidad de bits, y dado un voltaje de referencia de 5V, la resolución del ADC sería:

$$Resolución = \frac{5}{2^3 - 1} = 0.7142\,V.$$

En la tabla 1 se muestran los valores discretos de voltaje que el ADC entonces sería capaz de representar:

Tabla 1. Niveles de voltaje representados por un ADC de 3 bits con V_{ref} = 5V

Combinación binaria a la salida	**Equivalente decimal**	**Nivel de voltaje representado**	**Combinación binaria a la salida**	**Equivalente decimal**	**Nivel de voltaje representado**
000	0	0 V	100	4	2.8571 V
001	1	0.7142 V	101	5	3.5714 V
010	2	1.4285 V	110	6	4.2857 V
011	3	2.1418 V	111	7	5 V

Alternativamente, con una profundidad de 16 bits y un voltaje de referencia de 5V, se podrían representar 65,536 valores discretos de voltaje desde 0V a 5V, resultando en una resolución de 76.2951 µV, y por lo tanto una precisión mucho mayor.

La precisión del ADC dependerá del periodo de muestreo, que limita su precisión en el tiempo, y la profundidad de bits, que limita su precisión en magnitud, como se muestra en la figura 47. Estos dos parámetros juntos forman una malla imaginaria que muestra todas las posiciones posibles donde se pueden colocar las muestras.

Figura 47: Digitalización de una onda con profundidad de 3 bits y periodo de muestreo igual a 1/8 el periodo de la onda

2.4.2.1 Efecto alias

Es la obtención de una señal con una frecuencia diferente a la de una señal muestreada, causado por llevar a cabo el muestreo con una tasa de muestreo baja relativo a la frecuencia de oscilación de la señal analógica, representado gráficamente en la figura 48.

En la figura 49 se muestra la frecuencia que se obtendrá a partir de la señal original con una tasa de muestreo f_s. En la región de $0.5f_s$ a $1.5f_s$, la frecuencia obtenida se puede calcular a través de la siguiente ecuación:

$$f_{obtenida} = |f_s - f_{original}| \tag{30}$$

Entonces, sea la figura 48 un caso donde la onda azul tiene una frecuencia de 440 Hz, y se utiliza una tasa de muestreo de 380 Hz, la frecuencia de la onda medida sería:

$$f_{obtenida} = |380\ Hz - 440\ Hz| = 60\ Hz$$

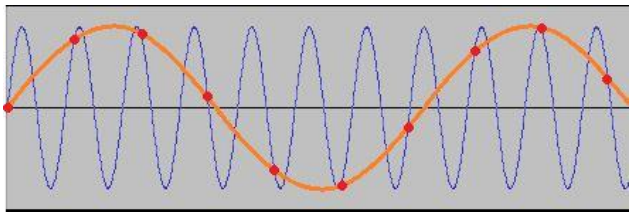


Figura 48: Visualización de efecto alias, donde la onda azul es la onda original, los puntos rojos son las muestras, y la onda anaranjada es la onda medida

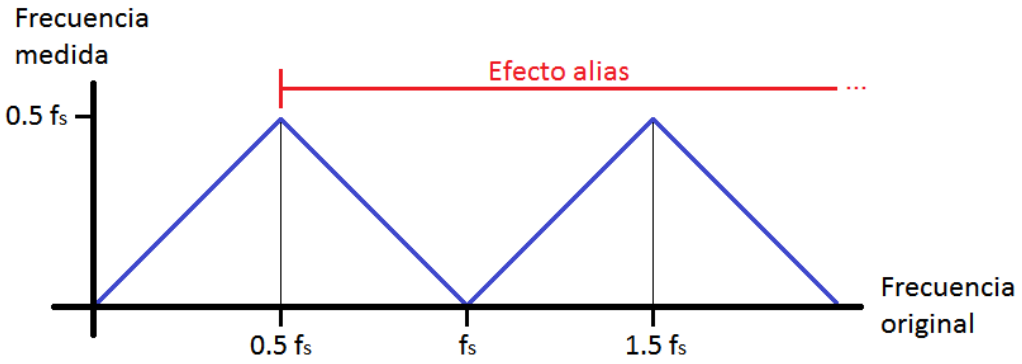


Figura 49: Gráfica de las frecuencias medidas en función de la frecuencia original y la tasa de muestreo f_s

Para evitar este efecto, se recomienda muestrear con una tasa de al menos el doble de la frecuencia más alta presente en la señal analógica. A esta recomendación se le conoce como Teorema de Nyquist, representado como:

$$f_s \geq 2\, f_{máx} \tag{31}$$

donde $f_{máx}$ es la frecuencia máxima presente en la señal original.

Aunque cumplir con el Teorema de Nyquist garantiza la obtención de la frecuencia correcta, no garantiza que la forma de onda como tal vaya a ser precisa. Para ello, usualmente se recomienda muestrear con una tasa de muestreo al menos 10 veces el de la frecuencia máxima de la señal.

Esta es una consideración importante de mantener en cuenta, pues si se va a medir una señal utilizando un dispositivo digital, como la tarjeta de sonido de una computadora, un micrófono USB, o un osciloscopio digital, habrá que asegurarse que la frecuencia de muestreo sea suficiente para la aplicación, por ejemplo en el caso de querer validar la operación correcta del sintetizador propuesto en esta tesis.

Para música digital, se utilizan frecuencias de muestreo mayores a 40 kHz, típicamente 44.1 kHz y 48 kHz, para asegurar que no ocurra efecto alias en el rango audible (menor a 20 kHz).

2.4.3 Conversión digital a analógico

El convertidor digital-a-analógico (DAC) comparte algunas características con el ADC. Por ejemplo, también contarán con una frecuencia de muestreo, aunque en este caso, es la frecuencia con la que es capaz de convertir muestras a un voltaje analógico, y una resolución de bits, que correspondería a los niveles discretos de voltaje que es capaz de producir. En la figura 50 se muestra un DAC simple de 4 bits.

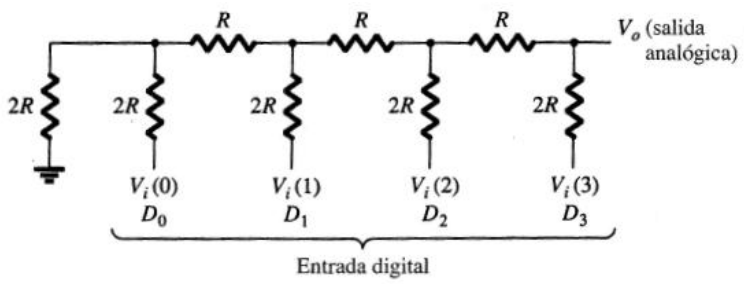


Figura 50: DAC a base de resistores (R. Boylestad y L. Nashelsky, Electrónica, pág. 718)

Al igual que en los ADC, la conversión digital-a-analógico puede tomar algunos instantes de tiempo, dependiendo del tipo de convertidor, y de si los bits del vector binario se reciben de forma paralela o por protocolo serial.

Realizar una sola conversión únicamente produce una señal constante, por lo que para producir una señal variante es necesario realizar conversiones consecutivas, actualizando los datos a la entrada del convertidor. Durante el tiempo entre una conversión y la siguiente, la salida del DAC se mantiene constante. Esto produce una forma de onda diferente a la esperada, y por ende ocasiona una distorsión, es decir, una alteración en los armónicos de la señal que se intenta producir, si las conversiones no son lo suficientemente rápidas.

Las señales eléctricas de audio oscilan alrededor de un valor 0 adoptando valores positivos y negativos de voltaje. Sin embargo, por la naturaleza binaria de los datos de entrada, y por el rango de voltajes disponibles desde 0 a V_{ref}, la onda a la salida se producirá con un *offset* tal que todos sus valores serán positivos, o cero en el caso del extremo inferior de oscilación. Esto se puede arreglar conectando la salida del DAC a un filtro analógico pasa-altas con una frecuencia de corte inferior a 20 Hz para eliminar la componente de corriente directa, al menos que se utilice un altavoz doméstico que requiera alimentación de corriente, pues estos usualmente ya cuentan con un amplificador y un filtro integrados.

El sintetizador propuesto en la presente tesis se programará en un dispositivo lógico programable, específicamente un FPGA, por lo que se procederá a describir estos dispositivos en las siguientes secciones.

2.4.4 Dispositivos lógicos programables

Un dispositivo lógico programable, o PLD (*programmable logic device*), es un dispositivo cuya estructura básica es una gran cantidad de arreglos de compuertas AND y compuertas OR, conectadas a las entradas y salidas del dispositivo, como se muestra en la figura 51. Estos arreglos cuentan con fusibles en las intersecciones que, mediante programación, se apagan eléctricamente, formando un circuito digital específico, y por ende una tabla de verdad a la salida.

La arquitectura de un PAL (*programmable logic array*, arreglo lógico programable), un tipo específico de PLD, está formada por circuitos combinacionales, a su vez formados por compuertas AND y OR, así como *buffers* e inversores.

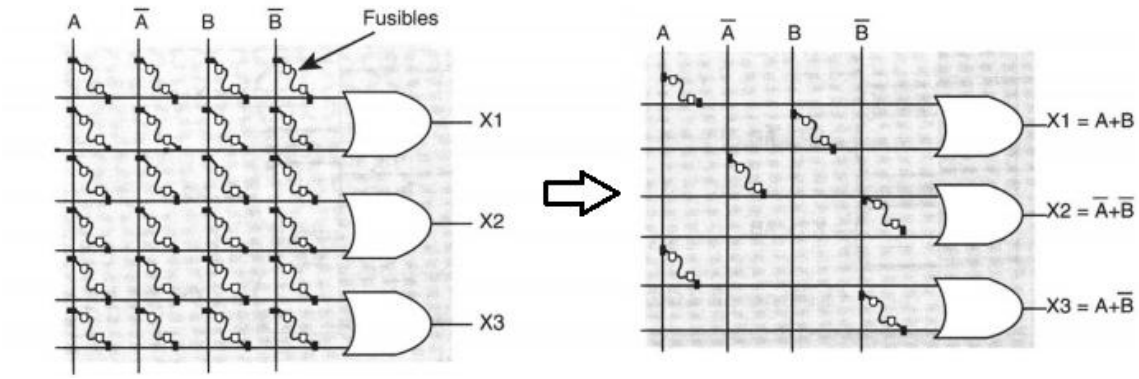


Figura 51: Estructura básica de un PLD antes y después de programar

2.4.4.1 FPGA

Un FPGA (*field-programmable gate array*, arreglo de compuertas programable en campo) es un dispositivo formado por bloques lógicos configurables (CLB, *configurable logic blocks*), bloques de entrada y de salida (IOB, *input-output blocks*) y canales de comunicación.

Los bloques lógicos configurables están formados por tablas de acceso (*lookup tables* o LUTs), que son arreglos programados de compuertas tal que producen una salida en función de una tabla de verdad, un sumador completo, y un flip-flop tipo D.

Del libro "VHDL: El arte de programar sistemas digitales" [30], pág. 27: "Los bloques lógicos (llamados también celdas generadoras de funciones) están configurados para procesar cualquier aplicación lógica. Estos tienen la característica de ser funcionalmente completos, es decir, permiten la implementación de cualquier función booleana representada en la forma de suma de productos. El diseño lógico se implementa mediante bloques conocidos como generadores de funciones o LUT, los cuales permiten almacenar la lógica requerida, ya que cuenta con una pequeña memoria interna. Cuando se aplica alguna combinación en las entradas de la LUT, el circuito la traduce en una dirección de memoria y envía fuera del bloque el dato almacenado en esta dirección."

La ventaja de los FPGA sobre los procesadores es que estos últimos ejecutan comandos secuencialmente, mientras que en un FPGA en realidad se están programando conexiones de hardware y por lo tanto las salidas son resultado de lógica combinacional, efectivamente generando salidas simultánea e instantáneamente.

Figura 52: Tarjeta Spartan 3E Starter Kit de Xilinx® (http://www.xilinx.com/products/boards-and-kits/hw-spar3e-sk-us-g.html)

Figura 53: FPGA XC3S500E de Xilinx® (https://www.sparkfun.com/products/retired/11657)

Los dispositivos FPGA se programan mediante lenguajes de descripción de hardware (HDL, *hardware description language*), tales como VHDL y Verilog. Las siglas VHDL significan "*VHSIC Hardware Description Language*". A su vez, VHSIC significa "*Very High Speed Integrated Circuit*" (circuito integrado de muy alta velocidad). Los comandos y la sintaxis como tales usados en la programación VHDL no están pre-establecidos a un formato único, sin embargo, es común utilizar el formato estándar 1076-2008 de la IEEE.

Los archivos de código HDL comúnmente son referidos como "módulos".

Los FPGA se pueden encontrar incluidos en tarjetas de desarrollo junto otros elementos tales como puertos de entrada y salida analógicos y digitales, LEDs indicadores, ADC, DAC, memoria flash, puertos para diversos conectores, etc.

El sintetizador desarrollado para la presente tesis se implementará en una tarjeta Spartan 3E Starter Kit (figura 52), que incluye el FPGA XC3S500E (figura 53). A continuación se enlistan algunas de las características de la tarjeta Spartan 3E:

- Frecuencia de reloj de 50 MHz
- *Push buttons*, *slide switches*, y LEDs integrados para prueba y depuración
- 12 puertos de entrada/salida de propósito general
- Conectores de diferentes tipos, entre ellos conector PS/2
- ADC LTC1407A-1 y DAC LTC2624
- Pantalla LCD

El FPGA XC3S500E contiene 9,312 LUTs de 4 entradas.

2.5 Síntesis de sonido

En esta sección se describirá brevemente qué es un sintetizador, además de algunos métodos de síntesis de sonido, con el fin de proporcionar una visión clara de la funcionalidad que se pretende implementar la presente tesis.

2.5.1 Sintetizadores

Un sintetizador es un instrumento musical electrónico con interfaz de teclado cuya principal característica es que le permite al usuario diseñar timbres variando parámetros como formas de onda, filtros, y osciladores para modulación. Algunos ejemplos se observan en las figuras 1, 2 y 3 (págs. 2 y 3).

Los sintetizadores contrastan con otros tipos de teclado musical, como los pianos digitales, que producen sonido por síntesis basada en muestras, por lo que los timbres que producen son predeterminados.

La arquitectura y funcionamiento de un sintetizador específico dependerá de su circuitería o software particular, y del método de síntesis para el que fue diseñado el sintetizador.

2.5.2 Algunos métodos de síntesis

Además de la síntesis por modulación de frecuencia, que es el método de síntesis que se utilizará en el sintetizador presentado en esta tesis, se explicarán las características básicas de la síntesis aditiva y la síntesis sustractiva. Es importante mantener en cuenta que existen otros métodos de síntesis de sonido además de éstos, tales como la síntesis basada en muestras, síntesis por modelado físico, síntesis granular, síntesis espectral, entre otros.

2.5.2.1 Síntesis aditiva

Es la suma de ondas estrictamente sinusoidales con el fin de construir una forma de onda específica. Se puede considerar, por ejemplo, que las series de Fourier llevan a cabo una síntesis aditiva.

- Ventajas: Ofrece la mayor flexibilidad para el diseño de sonidos. Se pueden imitar instrumentos reales si se conoce la amplitud y frecuencia de sus componentes espectrales.
- Desventajas: El número de osciladores disponibles puede estar limitado por el hardware. Para imitar un instrumento real, es necesario agregar una gran cantidad de componentes frecuenciales, por lo que la precisión de este método dependería del número de osciladores con el que cuente el sintetizador.

2.5.2.2 Síntesis sustractiva

Es simplemente la aplicación de un filtro pasa-bajas, variando su grado de filtrado, sobre un tono con alto contenido espectral. En sintetizadores, este método comúnmente se lleva a cabo aplicando el filtrado a una onda cuadrada, triangular, o diente de sierra.

- Ventajas: De los 3 métodos presentados, es el más sencillo de entender y utilizar.
- Desventajas: No existe mucha flexibilidad para el diseño de sonidos.

2.5.2.3 Síntesis por modulación de frecuencia

Se basa en el uso de una onda moduladora, cuya frecuencia se encuentra dentro del rango audible, para modificar la frecuencia de una onda portadora.

Si la onda moduladora tiene una frecuencia por debajo del rango audible (20 Hz), se percibiría un efecto de *vibrato*. Sin embargo, al tener la onda moduladora una frecuencia mayor a 20 Hz, se percibe una modificación en el timbre de la onda portadora en vez de un efecto de *vibrato*.

Se producen timbres más agradables cuando hay una proporción entera entre la frecuencia de la onda moduladora y la portadora, mientras que al haber una proporción no entera entre ellas se producirán componentes inarmónicas, y por lo tanto, un timbre disonante (es decir, percibido como “desagradable”).

A pesar de referirse a este método de síntesis de audio como síntesis por modulación de frecuencia, es posible producir un resultado equivalente por medio de modulación de fase cuando la onda moduladora es senoidal, como se explicó en la sección 2.1.3.

Un ejemplo es el sintetizador Yamaha DX7, que es comercializado como un sintetizador por modulación de frecuencia, aunque la síntesis realmente la hace por modulación de fase [32].

- Ventajas: Es posible crear sonidos complejos y con bastante contenido espectral utilizando únicamente dos osciladores. Adicionalmente, es posible conectar varios osciladores de onda moduladora en serie y/o paralelo para obtener sonidos aún más complejos.
- Desventajas: Su uso puede ser complicado para principiantes y requiere que se conozcan términos y conceptos técnicos de modulación.

Una vez revisados los conceptos relevantes matemáticos, físicos y musicales, así como de electrónica analógica y digital, es posible proponer un método para diseñar un sintetizador con un dispositivo digital, tal que se pueda interactuar con él bajo un contexto musical. A continuación se presenta la funcionalidad general propuesta.

3 Concepto del proyecto

Se propone diseñar, con un FPGA, un sintetizador por modulación de frecuencia, es decir, que sea capaz de producir una señal digital correspondiente a una onda sinusoidal modulada en frecuencia, que posteriormente se convertirá a una señal analógica por medio de un DAC.

El método de síntesis se basa en el sintetizador Yamaha DX7, es decir, ambas señales serán ondas sinusoidales y realmente se estaría llevando a cabo la síntesis mediante modulación de fase, produciendo una salida equivalente a una modulación de frecuencia.

Para los propósitos de la presente tesis, el sintetizador será monofónico, es decir, únicamente podrá reproducir una sola nota musical al mismo tiempo.

El sintetizador se programará en una tarjeta Spartan 3E de Xilinx®, y se utilizará un teclado de computadora que utilice el protocolo PS/2 para proporcionar instrucciones al FPGA, específicamente la reproducción de notas musicales, la selección de la octava musical, y la modificación de la amplitud y frecuencia de la onda moduladora. Estos parámetros se manipularán como se especifica a continuación:

- Nota musical: Se simulará una interfaz de teclado musical en un teclado de computadora, utilizando el renglón de la letra "A" para las teclas blancas, empezando por la nota "Do" en la tecla "A", y el reglón de la tecla "Q" para las teclas negras, empezando por la nota "Do#" en la tecla "W". El FPGA generará una onda sinusoidal de una frecuencia específica de acuerdo a la tecla presionada, únicamente mientras se mantenga presionada la tecla. La nota musical se interrumpirá cuando se deje de presionar la tecla.

- Octava musical: Se establecerá presionando cualquiera de las teclas numéricas correspondientes a los números desde el 0 al 7, que corresponderán directamente a la octava musical elegida. Es decir, si en el teclado se presiona la tecla correspondiente al número 4, el sintetizador producirá tonos musicales correspondientes a la 4ta octava al presionar las teclas asignadas a notas musicales.

- Frecuencia de la onda moduladora: Esta frecuencia influirá en la naturaleza de los armónicos introducidos, y por lo tanto en el timbre percibido. Se definirá como un múltiplo o submúltiplo de la frecuencia de la portadora (para retener el mismo timbre independientemente de la nota musical producida) y se modificará mediante las flechas izquierda y derecha del teclado.

- Amplitud de la onda moduladora: Modifica el rango de variación de frecuencia que la señal moduladora produce en la onda portadora. Esto también tiene un efecto sobre el timbre del tono musical, aunque es distinto al efecto producido por la modificación de la frecuencia de la onda moduladora. Este parámetro se modificará con las flechas arriba y abajo, produciendo un incremento o decremento en su amplitud por un factor de 2, o deshabilitando la modulación por completo.

La onda digital modulada se enviará al LTC2624 incluido en la tarjeta Spartan 3E. Debido a que este DAC recibe los datos por protocolo serial SPI, se incluirá un módulo VHDL que sea capaz de enviar el vector de bits a través de este protocolo.

La salida del DAC se conectará directamente a un altavoz doméstico con filtro de acoplamiento y amplificador integrado, evitando así la necesidad de armar manualmente un circuito para desacoplar la componente de corriente directa y un amplificador de potencia para la corriente.

En el diagrama a bloques de la figura 54 se muestra el funcionamiento a alto nivel del sintetizador propuesto.

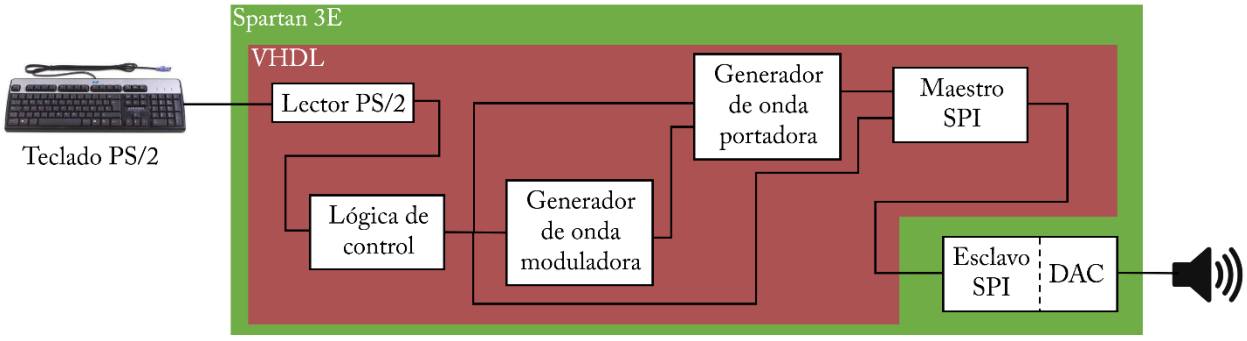


Figura 54: Diagrama a bloques del sintetizador FM implementado en FPGA

4 Desarrollo

Para la presente aplicación específica, la síntesis de ondas senoidales digitales se llevó a cabo mediante el módulo LogiCORE™ IP DDS (*Direct Digital Syntheziser*, Sintetizador Digital Directo) Compiler, que es un módulo VHDL compatible con diversos FPGA de Xilinx®, y que está disponible a través del CORE Generator System de Xilinx®.

El incremento de fase, descrito en la hoja de especificaciones (*datasheet*) del DDS Compiler y nombrado por la arquitectura específica del módulo, es el valor de entrada que definirá la frecuencia de la onda de salida, mediante la siguiente ecuación:

$$f_{out} = \frac{f_{clk}\Delta\theta}{2^{B_{\theta(n)}}} \tag{32}$$

donde f_{out} es la frecuencia de salida, f_{clk} es la frecuencia de reloj del sistema, $\Delta\theta$ es el incremento de fase, y $B_{\theta(n)}$ es la anchura de fase.

La anchura de fase se decide libremente y define la resolución de frecuencia, es decir, el mínimo incremento posible de frecuencia. Por lo tanto, la anchura de fase elegida debe resultar en una resolución de frecuencia suficiente para la aplicación, al mismo tiempo evitando que sea excesivo y requiera demasiado hardware.

La resolución de frecuencia se calcula mediante la siguiente ecuación:

$$\Delta f = \frac{f_{clk}}{2^{B_{\theta(n)}}} \tag{33}$$

En el rango musical decidido, el incremento mínimo de frecuencia requerido es de aproximadamente 1 Hz (Do0 = 16.3516 Hz a Do#0 = 17.3239 Hz). Sin embargo, debido al incremento exponencial de las frecuencias de las notas musicales, la diferencia de frecuencia aumenta gradualmente conforme se avanza a notas más agudas, dejando de ser múltiplos de 1 Hz. Por ello se buscó una anchura de fase que resulte en una resolución de frecuencia por lo menos menor a 0.1 Hz.

Despejando la ecuación 33, obtenemos el mínimo de bits requeridos para una resolución de frecuencia de 0.1 Hz:

$$B_{\theta(n)} = \left\lceil log_2\left(\frac{f_{clk}}{\Delta f}\right)\right\rceil = \left\lceil log_2\left(\frac{50\ MHz}{0.1\ Hz}\right)\right\rceil = \lceil 28.89 \rceil\ bits = 29\ bits$$

En base a este resultado, se decidió un ancho de fase de 32 bits para asegurar más de resolución, que en este caso sería:

$$\Delta f = \frac{f_{clk}}{2^{B_{\theta(n)}}} = \frac{50\ MHz}{2^{32}} = 0.01164\ Hz$$

Para ingresar la anchura de fase en el módulo DDS Compiler, en la primer página de la configuración se seleccionó "Parameter selection" > "Hardware parameters", y se ingresó "32" en "Phase Width". En esta misma pantalla, se ingresó "12" en "Output Width" para asegurar que la salida sea compatible con el LTC2624, que recibe vectores de datos de 12 bits.

Adicionalmente, el módulo se configuró para contar con entradas para el incremento de fase *PINC_IN* (seleccionando "Streaming" para "Phase increment programmability"), el *offset* de fase *POFF_IN* (seleccionando "Streaming" para "Phase offset programmability") que se usará para la acción de modulación de fase, y el habilitador *CE* (habilitando "Clock enable") para reproducir e interrumpir las notas musicales. Para la salida se habilitó el puerto *SINE*, donde se producirá el vector de bits que representa la amplitud instantánea de la onda digital modulada.

Se instanciaron 2 copias del módulo DDS Compiler, uno para la onda moduladora, y otro para la onda portadora. Para llevar a cabo la modulación, se conectó la salida del sintetizador digital de onda moduladora a la entrada *POFF_IN* del sintetizador digital de onda portadora. El rango digital de esta entrada equivaldría un rango de desfase de -180° a +180°.

Sin embargo, hay que considerar que la onda moduladora es de 12 bits, mientras que la entrada *POFF_IN* es de 32 bits. Para hacer ambos puertos compatibles, se concatenaron los 12 bits de la onda moduladora con 20 ceros en los bits menos significativos, siendo este el vector completo a la entrada de *POFF_IN*.

Por la cantidad de bits asignados a la salida de la onda modulada, este puede adoptar valores digitales desde 0 hasta 4095. Al soltar una tecla musical, el sintetizador digital de onda portadora se pausa únicamente hasta encontrarse dentro de un margen de ±5% del valor medio, es decir, entre 1946 y 2150, que serían cercanos a cero una vez eliminado el *offset*.

Es importante observar que el módulo DDS Compiler genera el semiciclo positivo de la onda en la mitad inferior de la salida, y el semiciclo negativo en la mitad superior de la salida, produciendo una onda como la observada en la figura 55. Para producir las ondas correctamente, simplemente se pasó el bit más significativo de cada onda por una compuerta NOT, y esta salida se concatenó con los otros 11 bits inalterados de la onda.

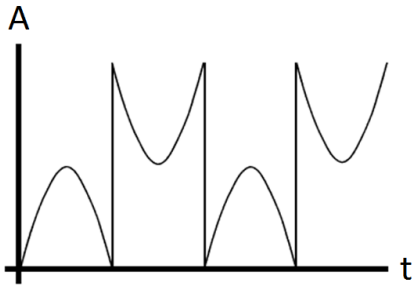


Figura 55: Salida del DDS Compiler con los semiciclos invertidos

La frecuencia con la que se genera la onda portadora, y por lo tanto, la nota musical que se reproduce, dependerá de la tecla presionada en el teclado de computadora. En determinado momento, se tendrá acceso a las notas musicales desde Do en una cierta octava, hasta Fa de la siguiente octava. En la figura 56 se muestran las teclas asignadas a cada nota musical, en el caso específico de utilizar un teclado de computadora con distribución el inglés.

Para contar con un rango musical amplio, se diseñó un esquema donde se únicamente se utilizarán las frecuencias de las notas musicales más graves (mínimo una octava completa) multiplicados por 2 elevado a un exponente variable manipulado por el usuario, para obtener las frecuencias de las notas en octavas posteriores. De esta forma sólo se requiere introducir en el código los valores de incremento de fase correspondientes a las frecuencias de una cantidad pequeña de notas musicales para tener acceso a un rango musical extenso.

Se decidió comenzar el extremo grave del rango musical en la octava 0. Utilizando la ecuación 19 (pág. 23) se calcularon las frecuencias de las notas musicales desde Do0 hasta Fa1, por ser el rango que permite el teclado de computadora. Aunque las notas antes de La0 se encuentren fuera del rango de un piano de 88 teclas, es posible calcular sus frecuencias. Por ejemplo, Do0 se consideraría la tecla número -8. Las frecuencias calculadas se muestran en la tabla 2.

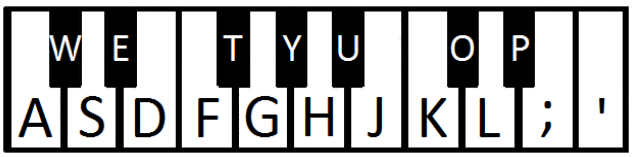


Figura 56: Notas musicales asignadas a diferentes teclas del teclado de computadora

Tabla 2. Frecuencias correspondientes a las notas musicales desde Do0 hasta Fa1

Nota musical	Frecuencia (Hz)	Nota musical	Frecuencia (Hz)
Do0	16.3516	La0	27.5000
Do#0	17.3239	Si♭0	29.1352
Re0	18.3540	Si0	30.8677
Mi♭0	19.4454	Do1	32.7032
Mi0	20.6017	Do#1	34.6478
Fa0	21.8268	Re1	36.7081
Fa#0	23.1247	Mi♭1	38.8909
Sol0	24.4997	Mi1	41.2034
La♭0	25.9565	Fa1	43.6535

Hay que tener en cuenta que las frecuencias debajo de 20 Hz son inaudibles. Sin embargo, se decidió comenzar desde la octava cero para poder incluir la nota La0, que es la nota más grave en un piano de 88 teclas, y además habilitar la posibilidad de explorar el caso de generación de armónicos audibles a partir de una frecuencia fundamental inaudible.

El incremento de fase necesario para generar una frecuencia de salida deseada se obtiene despejando la ecuación 32, dándonos:

$$\Delta\theta = \frac{f_{out} \cdot 2^{B_{\theta(n)}}}{f_{clk}} \tag{34}$$

Para las notas de la tabla 2, los incrementos de fase necesarios se calcularon utilizando esta ecuación, y se muestran en la tabla 3.

Sin embargo, los números resultantes deben ser ingresados en formato binario entero, requiriendo su conversión a número binario y descartando la porción decimal. Por lo tanto, los valores de incremento de fase ingresados al código fueron los de la tabla 3 redondeados a su entero más próximo. Esto causa un pequeño error en las frecuencias que realmente producirá el sintetizador, el cual se analizará más adelante. Estas nuevas frecuencias se pueden obtener ingresando los valores redondeados de incremento de fase en la ecuación 32, y se muestran en la tabla 4.

Tabla 3. Incrementos de fase correspondientes a las notas musicales desde Do0 hasta Fa1

Nota musical	Incremento de fase	Nota musical	Incremento de fase
Do0	1404.59	La0	2362.23
Do#0	1488.11	Si♭0	2502.70
Re0	1576.60	Si0	2651.52
Mi♭0	1670.35	Do1	2809.18
Mi0	1769.67	Do#1	2976.23
Fa0	1874.90	Re1	3153.20
Fa#0	1986.39	Mi♭1	3340.70
Sol0	2104.51	Mi1	3539.35
La♭0	2229.65	Fa1	3749.81

Tabla 4. Incrementos de fase redondeados y sus frecuencias resultantes

Incremento de fase redondeado	Frecuencia resultante (Hz)	Incremento de fase redondeado	Frecuencia resultante (Hz)
1405	16.3564	2362	27.4973
1488	17.3226	2503	29.1388
1577	18.3587	2652	30.8733
1670	19.4414	2809	32.7011
1770	20.6055	2976	34.6452
1875	21.8279	3153	36.7058
1986	23.1201	3341	38.8944
2105	24.5054	3539	41.1994
2230	25.9606	3750	43.6557

Las notas musicales en las octavas posteriores se obtendrán multiplicando los incrementos de fase mostrados en la tabla 4 por 2^n , donde el valor de ‘n’ será asignado directamente a través de las teclas numéricas del 0 al 7 (las encontradas encima de las teclas alfabéticas). Es decir, al presionar 0, los incrementos de fase se multiplican por 2^0, dejando las notas musicales en la octava 0 (agregando las notas excedentes de la siguiente octava por causa de la interfaz, que son Do1 a Fa1), mientras que presionar 4

ocasiona que los incrementos de fase se multipliquen por 2^4, dando como resultado las frecuencias de la 4ta octava (agregando las notas excedentes, en este caso, Do5 a Fa5).

Se eligió utilizar el 7 como la octava máxima para para que el Do más agudo incluido sea Do8, la nota musical más aguda en un piano de 88 teclas. Así, el rango total de notas musicales que el sintetizador es capaz de producir es desde Do0 hasta Fa8, cubriendo el rango de un piano de 88 teclas, más algunas notas adicionales en ambos extremos del espectro musical, para un total de 102 notas musicales.

En un vector binario, cada desplazamiento de los bits hacia la izquierda corresponde a una multiplicación por 2. Se aprovechó esta propiedad para lograr incrementos de octava fácilmente, únicamente desplazando los bits en la representación binaria de los incrementos de fase.

En una versión anterior del código, se utilizaron frecuencias de moduladora fijas, en vez de la propuesta actual donde la frecuencia de moduladora se define como un submúltiplo o múltiplo de la frecuencia portadora. Debido a que el timbre de una nota musical depende de la proporción de frecuencia y amplitud entre su fundamental y sus demás componentes, cambiar la frecuencia de portadora (al tocar diferentes notas musicales) dejando fija la frecuencia de moduladora ocasiona una variación en el timbre dependiendo de la nota musical específica que se toque.

Específicamente, esto genera componentes espectrales inarmónicas, por lo cual se percibiría un timbre ruidoso, disonante, y/o con notas musicales perceptibles diferentes a la que realmente se está tocando en el teclado. Por estas razones, este caso es indeseable, y al final se decidió trabajar con frecuencias de moduladora proporcionales a la frecuencia de portadora que se esté reproduciendo en el momento.

Sea f_c la frecuencia portadora generada en determinado momento, los posibles valores de frecuencia moduladora se definieron como f_c/128, f_c/32, f_c/16, f_c/8, f_c/4, f_c/2, f_c y $2f_c$. Se recorrerán estos valores con las teclas izquierda y derecha, en ese orden, con posiciones enumeradas del 1 al 8.

Aunque la onda moduladora será representada por un vector de bits, se puede considerar su amplitud máxima como 1. Esta amplitud se podrá modificar con las flechas arriba y abajo, siendo 0, 0.25, 0.5 y 1 los posibles valores, enumerados del 1 al 4.

Al igual que en el caso de los incrementos de octava, los diferentes valores disponibles de frecuencia y amplitud de la onda moduladora se obtendrán mediante simples desplazamientos de bits.

Los valores predeterminados al encender el sintetizador serán la octava número 4, una frecuencia de moduladora igual a f_c, y una amplitud de moduladora igual a 1.

La octava actual (representable con 3 bits), la posición de frecuencia moduladora (representable con 3 bits), y la posición de amplitud (representable con 2 bits) se visualizarán en formato binario en la tarjeta a través de los 8 LEDs manipulables que incluye.

La onda resultante, una vez convertida al dominio analógico, estará descrita por la ecuación 13 (pág. 10), considerando una variación máxima de fase de 180 grados, y la amplitud de señal moduladora seleccionada en determinado momento.

La representación binaria sin decimales de los valores calculados de incremento de fase, así como el cálculo utilizado en este caso particular para las frecuencias de las octavas superiores, causarán una pequeña diferencia entre las frecuencias estándar de las notas musicales, y las frecuencias que el sintetizador será capaz de producir. Por ello, se realizó un análisis para determinar si esta diferencia se pudiera percibir por el oído humano en caso de ser reproducida sin modulación contra un tono de prueba.

Esto se llevó a cabo calculando la diferencia entre la frecuencia producida por los cálculos y la frecuencia estándar de acuerdo a la ISO, así como el error relativo en porcentaje y en cents (unidad explicada en la sección 2.2.2.3.1), para las 102 notas musicales que el sintetizador es capaz de producir, y comparando los resultados con la figura 35. Debido a que la onda portadora es la que está directamente relacionada con la onda audible producida, este análisis no se llevó a cabo para la onda moduladora.

El error relativo del sintetizador en porcentaje en cada nota musical se calculó utilizando la siguiente fórmula:

$$E_{\%} = \left|1 - \frac{frecuencia\ reproducida}{frecuencia\ estándar}\right| x\ 100\% \tag{35}$$

Mientras que el error en cents se calculó utilizando la siguiente fórmula:

$$E_{cents} = \left|1200 \cdot \log_2\left(\frac{frecuencia\ reproducida}{frecuencia\ estándar}\right)\right| \tag{36}$$

Alternativamente, se puede calcular ingresando el resultado de la ecuación 35 en la ecuación 21 (pág. 30).

El máximo error de frecuencia encontrado fue de 0.7309 Hz en la nota Sol7 (cuya frecuencia estándar es 3.136 kHz), mientras que el máximo error *relativo* fue de 0.0291%, equivalente a 0.5 cents, en la nota Do0.

Basado en estos resultados y en las diferencias apenas perceptibles de frecuencia descritas en la figura 35, los errores de frecuencia producidos por el sintetizador no deberían poder ser perceptibles.

Para la lectura del teclado PS/2, se integró un módulo VHDL (apéndice B), el cual almacena en un vector *ps2_code* el código recibido por comunicación serial, y proporciona una señal booleana *code_new* que indica si hay una transferencia en proceso, o si la transferencia ha concluido y el código recibido está listo para su utilización.

En cuanto a la tasa de conversión digital-a-analógico, la limitante de rapidez del DAC LTC2624 incluido en la tarjeta Spartan 3E se debe a su recepción de datos por protocolo SPI. Considerando transferencias seriales de 32 bits a una velocidad $f_{clk}/2 = 25\ MHz$, la tasa de muestreo sería:

$$f_s = \frac{f_{clkSPI}}{n_{bits}} = \frac{25\ MHz}{32\ bits} = 781.25\ kHz$$

De acuerdo al teorema de Nyquist (ecuación 31, pág. 45), la tasa de muestreo resultante es suficiente para la máxima frecuencia de portadora utilizada, igual a 5.578 kHz, e incluso es suficiente considerando frecuencias instantáneas aún superiores por efecto de la modulación.

El módulo VHDL utilizado para la transferencia de datos por protocolo SPI (apéndice C) cuenta con un modo de operación continuo, que resulta ser muy útil para el envío consecutivo de amplitudes instantáneas digitales de una señal variante. Sin embargo, este modo ocasiona que no se actualice la salida de selección de esclavo *ss_n*. Esto presenta un problema, pues esta señal, que va conectada a la terminal *dac_cs* del LTC2624, debe cambiar a 0 para que el DAC acepte un nuevo vector entrante, y debe cambiar a 1 para ejecutar el comando y actualizar la salida analógica con el vector adquirido.

Considerando que la señal *busy* del módulo es prácticamente una versión invertida de la señal de selección de esclavo, y que sí se actualiza incluso operando en el modo continuo, se optó por conectar la señal *busy* a una compuerta NOT, y conectar esta nueva salida a la terminal *dac_cs* del LTC2624. La salida *ss_n* se eliminó del módulo original.

La lógica de control desarrollada para la operación del sintetizador se puede representar como la máquina de estados mostrada en la figura 57. A continuación se enlista la relación entre las etiquetas utilizadas en la figura, y la tecla o señal que le corresponde:

- “en”: señal *wave_enable*
- “new”: señal *code_new*
- “code”: señal *ps2_code*
- “tecla de nota musical”: teclas mostradas en la figura 55
- “tecla numérica”: números del 0 al 7

Las condiciones para el cambio de estado se muestran con signo de interrogación, mientras que las salidas durante el cambio de estado se marcan entre paréntesis.

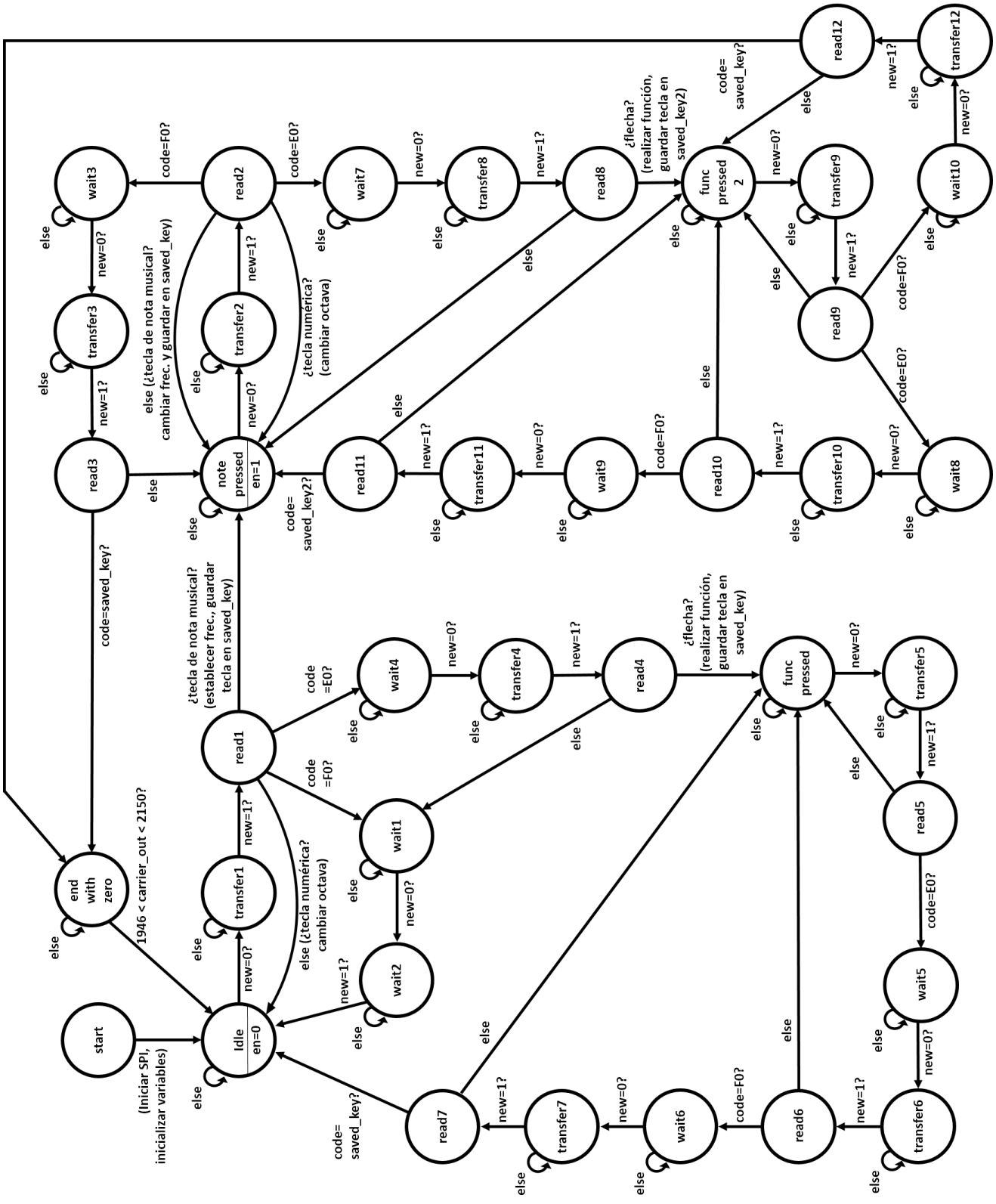


Figura 55: Máquina de estados desarrollada en VHDL

5 Experimento de validación

Para comprobar el correcto funcionamiento del sintetizador, se midieron las formas de onda y espectros de frecuencia generados por el sintetizador en 4 casos de operación utilizando un osciloscopio Tektronix® TDS1012C-EDU, y posteriormente, se utilizó MATLAB para generar manualmente ondas moduladas en fase con los mismos parámetros de frecuencia de portadora, frecuencia de moduladora, y amplitud de moduladora utilizados por el sintetizador en estos casos, con el fin de comparar los resultados experimentales con los simulados.

Además, se analizaron otros casos de operación auditivamente.

A continuación se presentan los resultados, especificando en cada caso la frecuencia portadora f_c, frecuencia moduladora f_m, y amplitud de moduladora A_m utilizados. Considérese 1 el valor máximo de A_m. En todos los casos se está reproduciendo la nota musical La4, cuya frecuencia es 440 Hz.

La amplitud en el tiempo está normalizada a la amplitud experimental, mientras que la magnitud en la frecuencia está normalizada al rango del osciloscopio.

- **<u>Caso 1</u>**: $f_c = 440\ Hz$
  $f_m = f_c/4 = 110\ Hz$
  $A_m = 1$

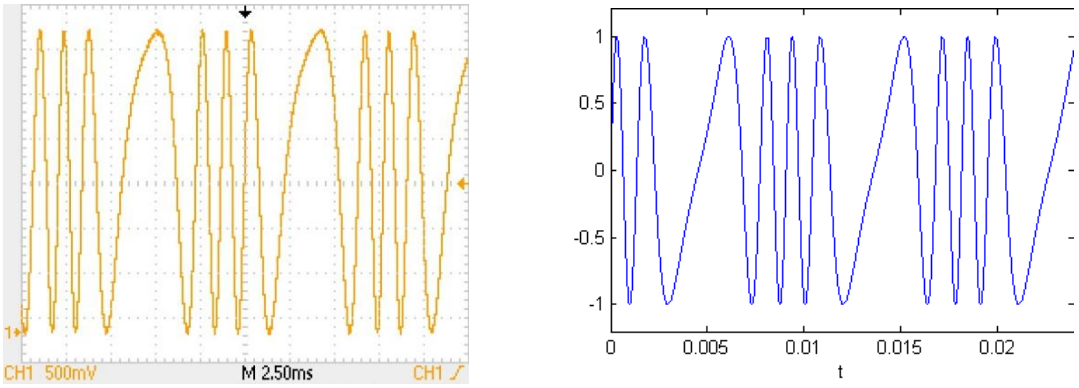


Figura 56: Comparación de formas de onda para caso 1 (izquierda: experimental, derecha: simulada)

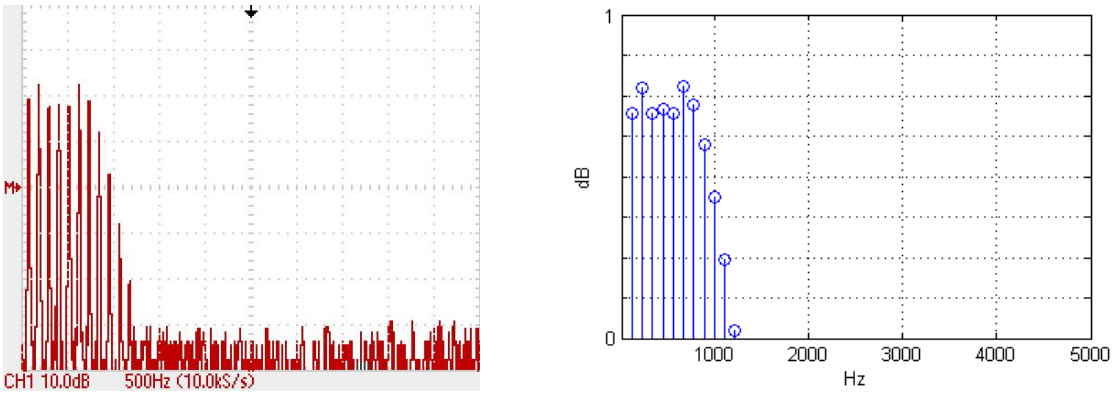


Figura 57: Comparación de espectros de frecuencia para caso 1 (izquierda: experimental, derecha: simulado)

- **Caso 2**: $f_c = 440\ Hz$
  $f_m = f_c/2 = 220\ Hz$
  $A_m = 1$

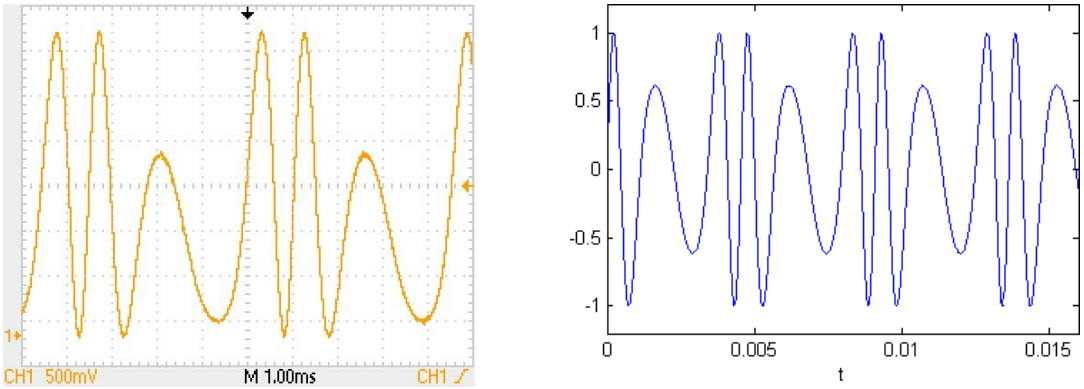


Figura 58: Comparación de formas de onda para caso 2 (izquierda: experimental, derecha: simulada)

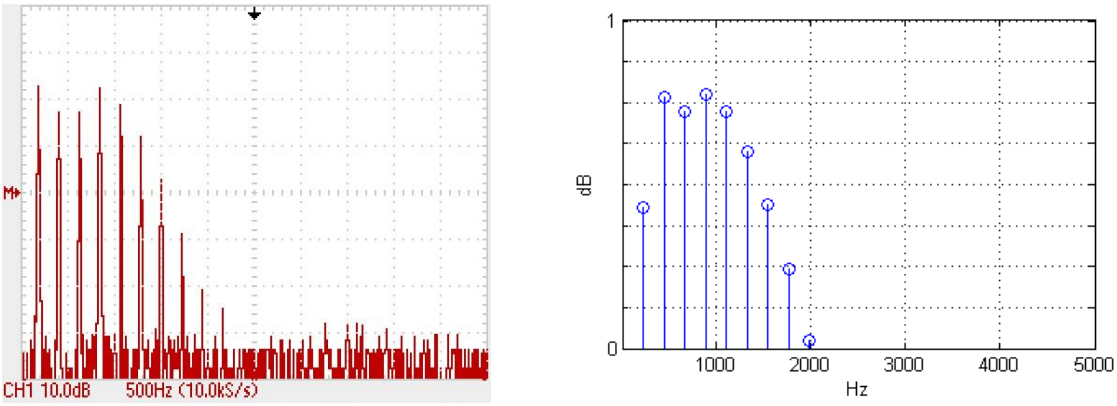


Figura 59: Comparación de espectros de frecuencia para caso 2 (izquierda: experimental, derecha: simulado)

- **Caso 3**: $f_c = 440\ Hz$
  $f_m = f_c = 440\ Hz$
  $A_m = 1$

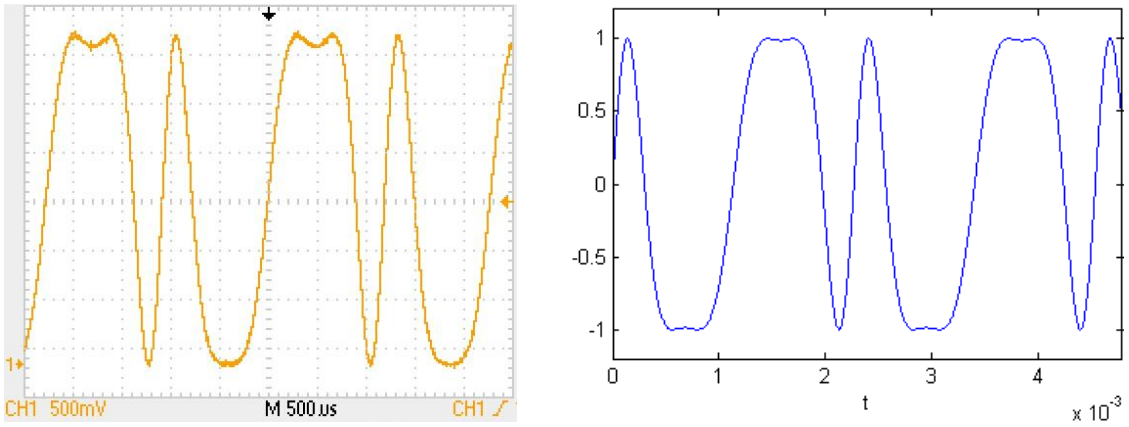


Figura 60: Comparación de formas de onda para caso 3 (izquierda: experimental, derecha: simulada)

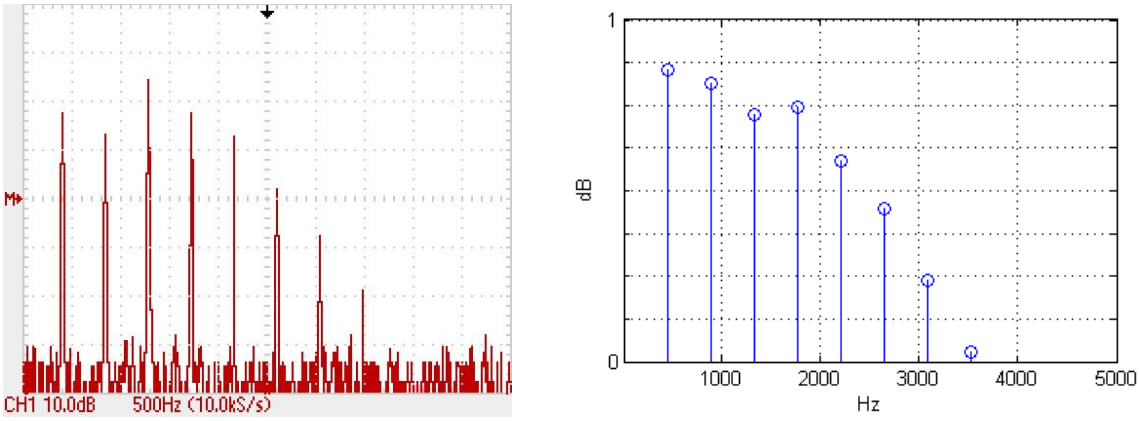


Figura 61: Comparación de espectros de frecuencia para caso 3 (izquierda: experimental, derecha: simulado)

- **Caso 4**: $f_c = 440\ Hz$
  $f_m = f_c = 440\ Hz$
  $A_m = 0.5$

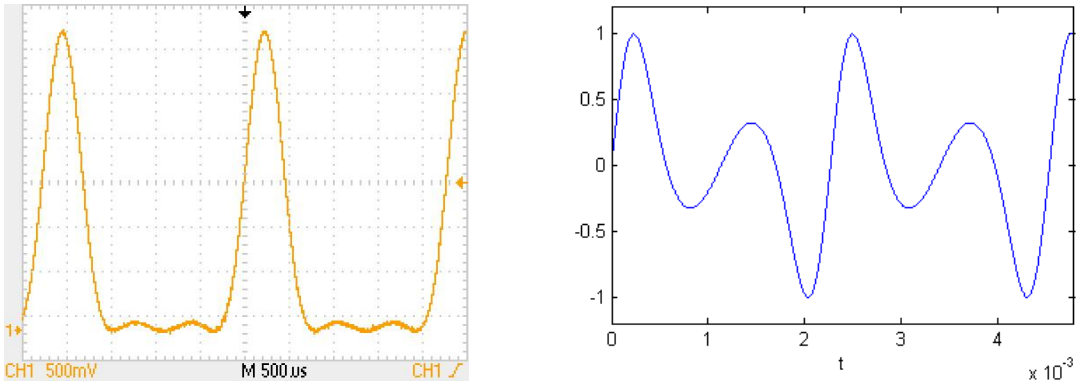


Figura 62: Comparación de formas de onda para caso 4 (izquierda: experimental, derecha: simulada)

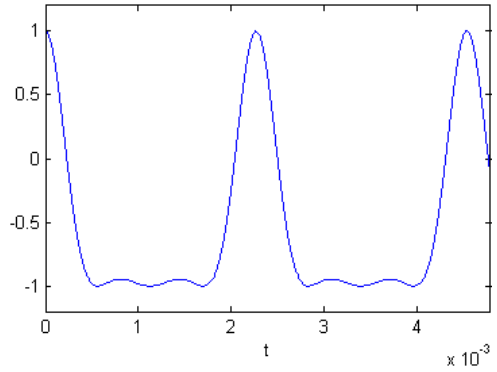


Figura 63: Forma de onda simulada para caso 4 con fase inicial de portadora de +90 grados

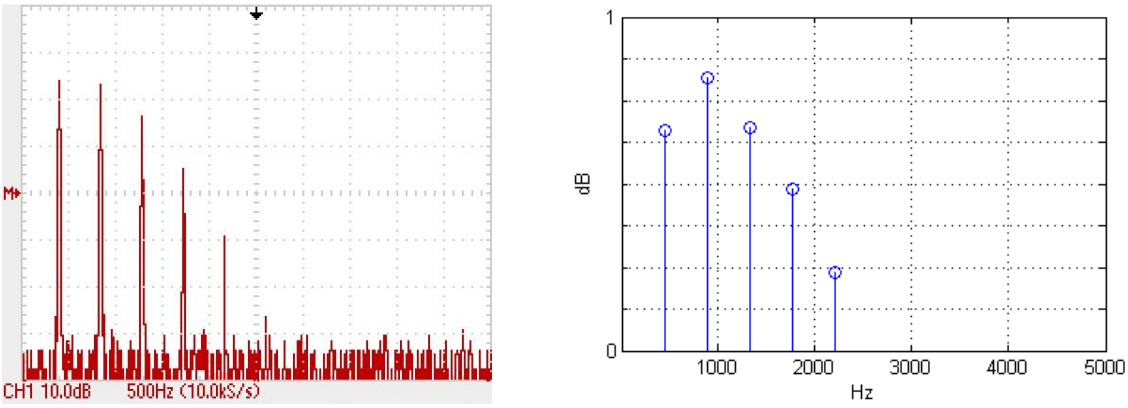


Figura 64: Comparación de espectros de frecuencia para caso 4 (izquierda: experimental, derecha: simulado)

6 Análisis de resultados

En los primeros 3 casos mostrados en la sección anterior sólo se observan variaciones ligeras en la forma de onda, mientras que en el caso 4, pareciera que la forma de onda no se está produciendo correctamente. Sin embargo, esto ocurre debido a una diferencia de fase entre la onda portadora y la moduladora. Estas diferencias de fase se generan durante la operación del sintetizador, al llevar a cabo incrementos y decrementos en la frecuencia moduladora, pues los cambios de frecuencia de moduladora pudieron haber sucedido en diferentes fases de la onda portadora. Las ligeras variaciones observadas en la forma de onda en los casos 1, 2 y 3, se pueden atribuir a este suceso.

Si se compensara por la diferencia en el caso 4 agregando a la onda portadora simulada una fase inicial de 90 grados, se obtiene la misma forma de onda que la medida experimentalmente, como se observa en la figura 65.

Adicionalmente, a pesar de la diferencia observada en el dominio del tiempo en este caso, se observa similitud en sus espectros de frecuencia mostrados en la figura 66, al igual que en los primeros 3 casos. Por lo tanto, considerando que el oído humano percibe el sonido principalmente en términos de su espectro de frecuencia, las ondas en el tiempo serían equivalentes para el propósito del presente trabajo, cuyo objetivo final es una percepción psicoacústica.

En los espectros de frecuencia se observan los efectos de modificar la frecuencia y la amplitud de la onda moduladora. La frecuencia de la onda moduladora influye en la separación entre las componentes espectrales, mientras que la amplitud de la onda moduladora influye en las amplitudes de las componentes sin desplazarlas en frecuencia.

Específicamente, se observa que la separación entre las componentes espectrales es igual a la frecuencia de modulación. Por ejemplo, en el caso 1, la frecuencia de moduladora es de 110 Hz, y las componentes espectrales se encuentran en 110 Hz, 220 Hz, 330 Hz, etc. De la misma manera, en el caso 2, con una frecuencia de moduladora de 220 Hz, se observan componentes en 220 Hz, 440 Hz, 660 Hz, etc.

Aumentar la amplitud de moduladora puede generar componentes a frecuencias más altas, pero únicamente a intervalos fijos de frecuencia determinados por la frecuencia de la moduladora, mientras que reducir la amplitud produce un efecto comparable al de un filtro pasa-bajas.

Hay que recordar que un tono musical está compuesto de una frecuencia fundamental, y armónicos cuyas frecuencias son mayores y proporcionales a la fundamental. Sin embargo, en este tipo de modulación se observa que se generan componentes frecuenciales más graves que la frecuencia de la onda

portadora. Esto ocasiona que, aunque se tenga una frecuencia de portadora de 440 Hz, correspondiente a la nota La4, el tono musical en el caso 1 se perciba como una nota musical La2, por ser 110 Hz su componente más grave, y las demás componentes múltiplos de 110 Hz.

Entre los casos analizados auditivamente, se logró comprobar la percepción de un efecto de *vibrato* con configuraciones que resultan en una frecuencia moduladora menor a 20 Hz, tal como teniendo una frecuencia de portadora de 220 Hz, y una frecuencia de moduladora $f_m = 220\,Hz/32 = 6.875\,Hz$. La magnitud del *vibrato* disminuye reduciendo la amplitud de la onda moduladora.

Utilizando la máxima frecuencia de moduladora permitida por el código, que es $2f_c$, el timbre percibido es parecido al de una onda cuadrada. Esto se debe a que, para una portadora de 440 Hz, la separación entre las componentes frecuenciales en este caso sería de 880 Hz, resultando en un espectro con componentes de 440 Hz, 1320 Hz ($3f_c$), 2200 Hz ($5f_c$), etc., y ya se había mencionado anteriormente que una onda cuadrada tiene componentes en f, $3f$, $5f$, etc., es decir, únicamente en sus múltiplos impares, a diferencia de la mayoría de los instrumentos musicales que tienen componentes tanto en sus múltiplos impares como los pares (f, $2f$, $3f$...)

Aun así, sigue habiendo una pequeña diferencia audible entre la onda modulada en frecuencia con $f_m = 2f_c$ y una onda cuadrada, debido a las diferencias de amplitud de las componentes de cada onda.

7 Conclusiones

Se diseñó, mediante código VHDL, un sintetizador musical FM a través de una técnica equivalente de modulación de fase, con un extenso rango musical total de 102 notas musicales (desde Do0 hasta Fa8), teniendo acceso en determinado momento a las notas entre una nota Do y la nota Fa de la siguiente octava. Además se cuenta con la habilidad de manipular la frecuencia y la amplitud de la onda moduladora, en intervalos exponenciales por factores de 2.

Las diferencias entre las frecuencias reproducidas por el FPGA y las frecuencias correspondientes a las notas musicales son suficientemente pequeñas para ser imperceptibles, siendo la mayor diferencia relativa producida por el FPGA 17 veces más pequeña que la menor diferencia relativa perceptible por el ser humano.

Por la naturaleza musical del proyecto, hubiera sido preferible manipular el sintetizador a través de un controlador MIDI con interfaz de teclado musical, sin embargo, no se contó con los recursos necesarios para llevar a cabo esta alternativa.

Por ser una implementación basada en código (disponible en el apéndice D), es sencillo ampliar la funcionalidad del sintetizador presentado en esta tesis, siendo éstas algunas de las oportunidades de mejora:

- Habilidad para reproducir más de una nota musical a la vez
- Control del sintetizador a través de un controlador MIDI con interfaz de teclado musical
- Mayor resolución para manipular la amplitud y frecuencia de la moduladora (factores de multiplicación y división diferentes a potencias de 2)
- Implementación de la pantalla LCD de la Spartan 3E para visualizar los parámetros actuales

Con las modificaciones apropiadas, es posible basarse en la lógica y la teoría presentadas en esta tesis, tales como el cálculo de frecuencias musicales, para desarrollar sintetizadores que operen bajo otros tipos de síntesis de sonido, como la síntesis aditiva y la síntesis sustractiva.

El trabajo desarrollado en la presente tesis, así como los resultados de la misma, me dejan con una gran satisfacción y sentido de realización, pues mi pasión es trabajar con música y sonido desde un punto de vista físico-matemático y de ingeniería, y estoy muy agradecido no sólo por todos los conocimientos de electrónica adquiridos en la carrera (que es tanto tal que a veces me parece un poco difícil de creer), sino además por la conocimiento científico que me ha permitido conocer más íntima y objetivamente a la naturaleza y al Universo, y por supuesto por la posibilidad de aplicar todo esto en las disciplinas de la ingeniería acústica y la música.

Referencias

[1] US patent 580,035, Thaddeus Cahill, "Art of and apparatus for generating and distributing music electrically", publicado 1897-04-06

[2] Glinsky, Albert (2000), Theremin: Ether Music and Espionage, Urbana, Illinois: University of Illinois Press, p. 26, ISBN 0-252-02582-2

[3] Rhea, Thomas L., "Harald Bode's Four-Voice Assignment Keyboard (1937)", eContact! (reprint ed.) (Canadian Electroacoustic Community) 13 (4) (July 2011), originalmente publicado como Rhea, Tom (December 1979), "Electronic Perspectives", Contemporary Keyboard 5 (12): 89

[4] Musicweb.ch, 'Hammond organ | Musicweb.ch', 2015. [En línea]. Disponible en: http://www.musicweb.ch/en/hammond-organ. [Consultado: 29- Nov- 2015].

[5] Bode, Harald (1961), "European Electronic Music Instrument Design", Journal of the Audio Engineering Society (JAES) ix (1961): 267

[6] 'Moog Minimoog Voyager | Vintage Synth Explorer', *Vintagesynth.com*, 2015. [En línea]. Disponible en: http://www.vintagesynth.com/moog/voyager.php. [Consultado: 29- Nov- 2015].

[7] 'Yamaha DX7 Vintage Digital Polyphonic FM Synthesizer', *Reverb.com*, 2015. [En línea]. Disponible en: https://reverb.com/item/479871-yamaha-dx7-vintage-digital-polyphonic-fm-synthesizer. [Consultado: 29- Nov- 2015].

[8] L. Kinsler, *Fundamentos de acústica*. México: Limusa Noriega Editores, 1999.

[9] J. Roederer, *The physics and psychophysics of music*. New York: Springer-Verlag, 1995.

[10] D. Havelock, S. Kuwano and M. Vorländer, *Handbook of signal processing in acoustics*. New York: Springer, 2008.

[11] S. Haykin, *Communication systems*. New York: Wiley, 1994.

[12] J. Proakis and M. Salehi, *Fundamentals of communication systems*. Prentice Hall, 2013.

[13] P. Panter, *Communication systems design*. New York [etc.]: McGraw-Hill Book Company, 1972.

[14] D. Blackstock, *Fundamentals of physical acoustics*. New York: Wiley, 2000.

[15] R Nave. "Transverse and longitunial waves" [en línea]. Disponible en: http://hyperphysics.phy-astr.gsu.edu/hbase/sound/tralon.html [Consultado: 28-Nov-2015]

[16] E. Goldstein, *Sensación y percepción*. México, D.F.: International Thomson, 1999.

[17] P. Guillaume, *Music and Acoustics, from Instruments to Computer*. Francia: Wiley-ISTE, 2006.

[18] E. Herrera, *Teoría musical y armonía moderna*. Barcelona, España: Antoni Bosch, 1995.

[19] YouTube, 'AwesomeAudioChannel', 2015. [En línea]. Disponible en: https://www.youtube.com/channel/AwesomeAudioChannel. [Consultado: 30- Nov- 2015].

[20] Iso.org, 'ISO - International Organization for Standardization', 2015. [En línea]. Disponible en: http://www.iso.org/. [Consultado: 29- Nov- 2015].

[21] D. Howard and J. Angus, *Acoustics and psychoacoustics*. Oxford: Focal Press, 2001.

[22] H. Fastl and E. Zwicker, *Psychoacoustics*. Berlin: Springer, 2007.

[23] V. Pulkki and M. Karjalainen, *Communication Acoustics*. Somerset: Wiley, 2014.

[24] D. Rocchesso, *Introduction to sound processing*. Verona: Università di Verona, 2003.

[25] Shureasia.com, 'Shure Asia|SM58 The legendary vocal microphone', 2015. [En línea]. Disponible en: http://www.shureasia.com/en/products/microphones/sm58-vocal-microphone. [Consultado: 29- Nov- 2015].

[26] Shure.co.uk, 'Beta 52A Kick Drum Microphone', 2015. [En línea]. Disponible en: http://www.shure.co.uk/products/microphones/beta_52a. [Consultado: 29- Nov- 2015].

[27] W. Tomasi y V. García Bisogmo, *Sistemas de comunicaciones electrónicas*. México: Prentice Hall, 1996.

[28] S. Mitra, *Procesamiento de señales digitales*. México: McGraw Hill, 2007.

[29] R. Boylestad y L. Nashelsky, *Electronica*. Medellín: Prentice Hall, 1983.

[30] D. Maxinez y J. Alcalá Jara, *VHDL*. México: Compañia Editorial Continental, 2002.

[31] Standards.ieee.org, 'IEEE-SA - The IEEE Standards Association - Home', 2015. [En línea]. Disponible: http://standards.ieee.org. [Consultado: 29- Nov- 2015].

[32] R. Boulanger, *The Csound book*. Cambridge, Mass.: MIT Press, 2000, pág. 266.

[33] P. Chu, *FPGA prototyping by VHDL examples*. Hoboken, N.J.: Wiley-Interscience, 2008.

[34] Xilinx, Spartan-3E Starter Kit Board User Guide, 2006.

Apéndice A. Sistemas numéricos decimal, binario, y hexadecimal

La base de un sistema numérico es el número máximo de valores que se pueden representar con un sólo dígito.

Por ejemplo, el sistema numérico comúnmente utilizado es base 10, denominado "sistema decimal", por poder representar hasta 10 valores diferentes en un sólo dígito, siendo estos del 0 al 9.

En todo sistema numérico, cada dígito adicional hacia la izquierda (en el dominio de los números enteros) tendrá un valor igual al dígito multiplicado por la base elevada a la posición del dígito.

Para el número decimal 136, por ejemplo, el primer dígito tendría el valor de $1x10^2 = 100$, el siguiente dígito tendría el valor de $3x10^1 = 30$, y el dígito menos significativo tendría el valor de $6x10^0 = 6$. Al sumar estos 3 valores, obtenemos el valor del número original.

136 →	1		3		6			
	$x10^2$		$x10^1$		$x10^0$		(sistema decimal)	
	100	+	30	+	6		$= 136$	

El sistema numérico binario cuenta con base 2, lo que significa que cada dígito puede tener un máximo de dos posibles valores: 0 y 1. Por lo tanto, en este caso, el valor de cada dígito se multiplicaría por 2 elevado a la posición del dígito.

En el caso del número binario 101 (representado como 101_2 para evitar confusión con el sistema decimal), el dígito del extremo izquierdo sería igual a $1x2^2 = 4$, el siguiente sería igual a $0x2^1 = 0$, y el dígito menos significativo sería igual a $1x2^0 = 1$. Al sumar estos tres valores, se obtiene 5. Por lo tanto, 101 en sistema binario es equivalente al número 5 en sistema decimal.

101_2 →	1		0		1			
	$x2^2$		$x2^1$		$x2^0$		(sistema binario)	
	4	+	0	+	1		$= 5$	

El sistema hexadecimal cuenta con base 16, por lo que cada dígito puede tomar uno de 16 valores. Los primeros 10 serían del 0 al 9, y los siguientes valores se representan con las letras de la A a la F, de modo que un dígito en sistema hexadecimal puede tener uno de estos valores, en orden ascendente:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, y F, donde la letra A equivaldría a un 10 decimal, la B equivaldría a un 11 decimal, etc.

Debido a que la base de este sistema es potencia de 2, la conversión de sistema binario a hexadecimal es sencilla, donde un número binario se puede separar en grupos de 4 dígitos, y cada grupo equivaldría a un sólo dígito hexadecimal, sin influir en dígitos hexadecimales anteriores o posteriores.

Por ejemplo, el número 101101011110_2 se puede visualizar de la siguiente forma:

1011	0101	1110	(binario)

y posteriormente se puede convertir al número hexadecimal B5E.

1011	0101	1110	(binario)
11	5	14	(decimal)
B	5	E	(hexadecimal)

Esto conlleva a que una aplicación común de los números hexadecimales sea representar números binarios utilizando menos caracteres. Un ejemplo es el “byte”, que es igual a ocho dígitos binarios, pero típicamente es representado con dos dígitos hexadecimales.

Apéndice B. Protocolo PS/2

Se utiliza principalmente para la comunicación serial síncrona proveniente de un teclado o ratón de computadora, aunque actualmente han sido desplazados por los teclados y ratones USB en la vida cotidiana.

Por la naturaleza de la presente tesis, únicamente se describirá la transferencia de información de un teclado.

La comunicación se lleva a cabo a través de una línea de datos, y una línea de reloj dedicada al protocolo, usualmente derivada de la señal de reloj del sistema externo. Esta línea de reloj se utiliza para sincronizar el transmisor con el receptor, asegurando la lectura correcta de los bits.

Mientras no haya comunicación, la línea de datos y la línea de reloj se mantienen en un nivel lógico alto. Al presionar o soltar una tecla, se envía un bit de inicio en nivel lógico bajo, lo que arranca el reloj de sincronización. A continuación, se mandan 8 bits correspondientes al mensaje enviado por el teclado, comenzando por el bit menos significativo. Después del último bit se envía un bit de paridad, y se finaliza la comunicación con un bit de detenimiento en nivel lógico alto. Esta cadena de bits se representa gráficamente en la figura 67.

Los 8 bits de datos que se envían corresponden al código hexadecimal asignado a cada tecla específica. Al presionar una tecla, se enviará un *make code* correspondiente a la tecla presionada, que se muestra en la figura 68. Aunque la mayoría de las teclas tienen asignadas un sólo byte, algunas teclas de función, como las flechas, tienen 2 códigos, el primero siendo el byte E0. En este caso, ambos códigos se enviarán consecutivamente, enviando el segundo byte 2.6 ms después de finalizar la transmisión del primero. Si se deja presionada una tecla, se enviará su *make code* repetidamente aproximadamente cada 100 ms.

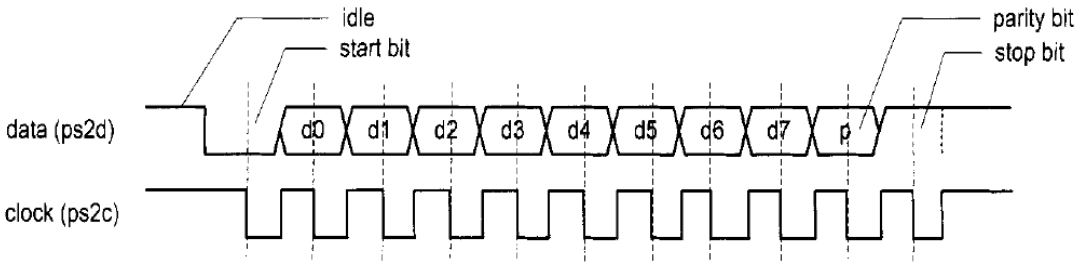


Figura 65: Diagrama de temporización del envío de un paquete de datos por protocolo PS/2

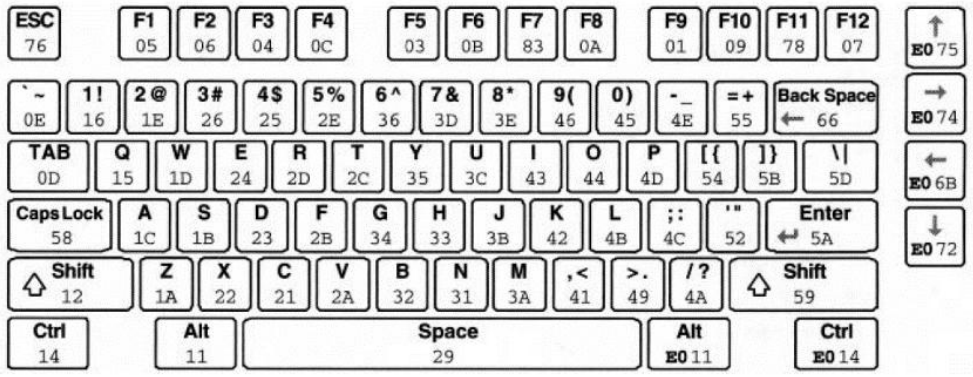


Figura 66: Códigos make asignados en un teclado

Al soltar una tecla presionada, se envía un *break code*. Este código usualmente consiste en el byte F0 seguido del *make code* de la misma tecla, aunque para el caso de teclas con el código E0, primero se envía E0, posteriormente F0, y finalmente el segundo byte del *make code* de la tecla.

Los códigos *make* y *break* de las teclas Print Screen y Pause son las únicas que no siguen este esquema.

Apéndice C. Protocolo SPI

El protocolo SPI (*Serial Peripheral Interface*, interfaz serial de periférico), es un protocolo de comunicación serial síncrono utilizado principalmente para la comunicación entre componentes cercanos de un mismo sistema.

Este protocolo utiliza un esquema de maestro/esclavo, como se muestra en la figura 69, donde el maestro proporciona la señal de reloj para la sincronización, y es capaz de seleccionar a cuál de varios esclavos conectados enviarle los datos. La cadena de bits se envía del maestro a un esclavo a través de la terminal "mosi" (*master out slave in*).

Existen diversos parámetros que pueden variar dependiendo del sistema, tales como la polaridad y fase de la señal de reloj, su frecuencia, el tamaño de la cadena de bits, etc.

Para el caso específico de una tarjeta Spartan 3E enviando datos al DAC LTC2624 integrado en la misma tarjeta, el tamaño de la cadena de bits es de 32. Se comienza con 8 bits despreciables, seguidos de 4 bits de comando. La interpretación de los bits de comando dependerá del dispositivo como tal. En el presente caso, el comando "0011" ejecuta una actualización inmediata de la salida del DAC. A continuación se envían 4 bits de dirección, con el que se elige en cuál de las 4 salidas del LTC2624 se actualizará el valor obtenido. Posteriormente se envían 12 bits correspondientes al vector binario que se desea convertir en una señal analógica. Se finaliza la transmisión con 4 bits despreciables. Esta transmisión se observa en la figura 70.

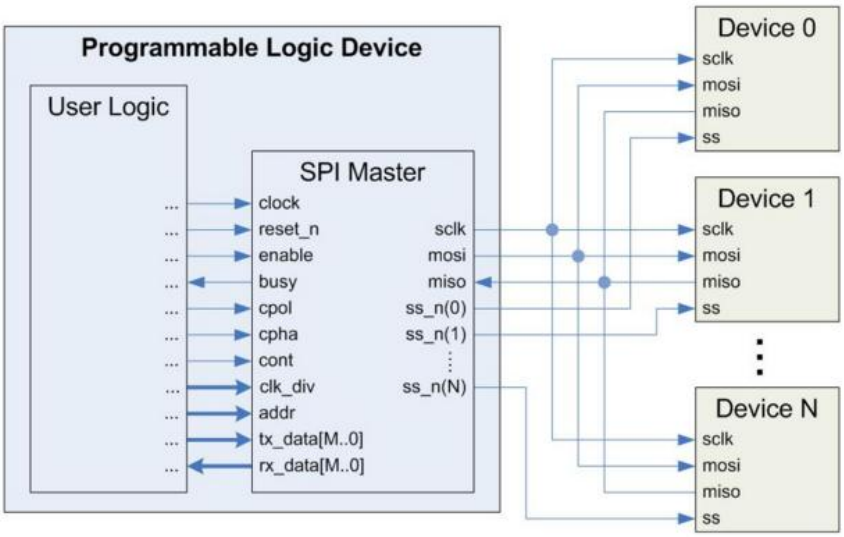


Figura 67: Ejemplo de esquema de comunicación SPI

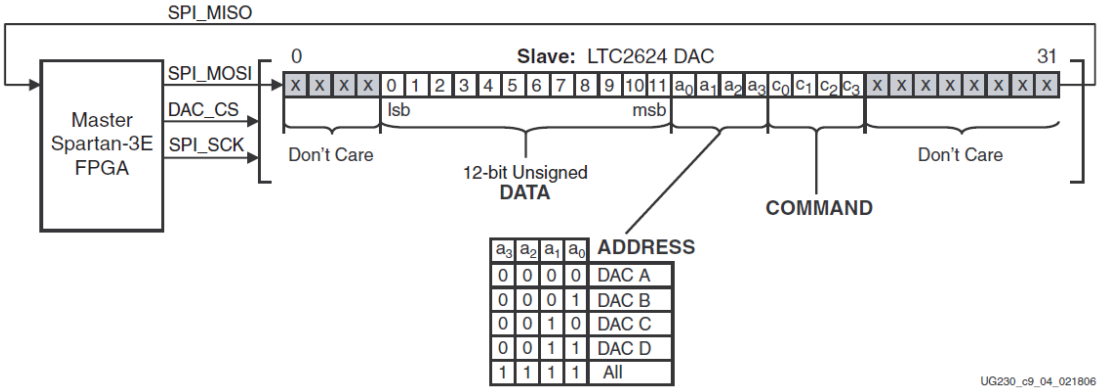


Figura 68: Comunicación SPI del FPGA al LTC2624

Apéndice D. Código VHDL

A continuación se presentará el código VHDL utilizado para la presente tesis, junto con sus fuentes en caso de haber sido adquiridos de internet.

El presente código fue implementado a la tarjeta Spartan 3E a través del software ISE Design Suite.

D.1 Módulo principal

Este módulo fue desarrollado en su totalidad por el ponente de la presente tesis, y principalmente lleva a cabo la lógica de control.

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    14:19:25 11/14/2015
-- Design Name:
-- Module Name:    pmsynthmain - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;



--pmsynthmain por Mario Alfredo Montoya Sandoval

--Teclas de nota musical en orden de escala cromática (Do, Do#, Re, Mib, Mi...):
--A W S E D F T G Y H U J K O L P ; '

--Cambiar octava: teclas numéricas (0, 1, 2, 3, 4, 5, 6, 7)

--Amplitud de moduladora (profundidad de modulación): Teclas arriba y abajo
```

```
--4 niveles (0, 1/4, 1/2, 1)

--Frecuencia de moduladora: Teclas izquierda y derecha
--8 niveles (frec nota/128, frec nota/32, frec nota/16, frec nota/8, frec nota/4,
--  frec nota/2, frec nota, frec nota*2),



entity pmsynthmain is
    Port ( clk : in  STD_LOGIC; --reloj de sistema
           reset : in  STD_LOGIC; --reset del sistema
           ps2_ck : in  STD_LOGIC; --reloj de teclado PS2
           ps2_data : in  STD_LOGIC; --datos seriales de teclado PS2
           carrier_octave_led : out  STD_LOGIC_VECTOR (2 downto 0); --LEDs indicadores de
octava de portadora
           mod_rate_led : out  STD_LOGIC_VECTOR (2 downto 0); --LEDs indicadores de
frecuencia de moduladora
                       mod_amp_led : out  STD_LOGIC_VECTOR (1 downto 0); --LEDs indicadores
de amplitud de moduladora
           spi_out : out  STD_LOGIC; --Datos seriales al SPI del DAC
           spi_sck : out  STD_LOGIC; --Reloj al SPI del DAC
           dac_cs : out  STD_LOGIC; --Chip select al SPI del DAC
           disable_others : out  STD_LOGIC_VECTOR (3 downto 0); --Para deshabilitar otros
SPI
           dac_clr : out  STD_LOGIC; --CLEAR al SPI del DAC
                       ps2_data_out: out STD_LOGIC); --Para medir (visualizar) código
serial PS/2
end pmsynthmain;

architecture Behavioral of pmsynthmain is

       signal code_new: std_logic; --bandera de nuevo código de teclado
       signal ps2_code: std_logic_vector (7 downto 0); --código PS/2 recibido
       signal ps2_data_signal: std_logic; --Nodo intermediario

       signal carrier_key: integer; --Número de nota de portadora
       signal carrier_note: integer; --Incremento de fase para notas de octava cero (antes
de aplicar octava)
       signal carrier_octave: integer; --Octava de portadora
       signal carrier_freq: std_logic_vector (31 downto 0); --Incremento de fase para notas
en la octava seleccionada

       signal mod_rate: integer; --Selección de frecuencia de moduladora
       signal mod_freq: std_logic_vector (31 downto 0); --Incremento de fase para
moduladora
       signal mod_amp: integer; --Amplitud de moduladora

       signal mod_amp_next, mod_rate_next: integer; --Para almacenar incrementos y
decrementos

       signal wave_enable: std_logic; --Habilitar/deshabilitar osciladores

       signal inv_carrier_out: std_logic_vector (11 downto 0); --Salida de generador de
portadora (semiciclos invertidos)
       signal carrier_out: std_logic_vector (11 downto 0); --Salida de generador de
portadora (semiciclos corregidos)

       signal inv_mod_out: std_logic_vector (11 downto 0); --Salida de generador de
moduladora (semiciclos invertidos)
       signal mod_scaled: std_logic_vector (31 downto 0); --Onda moduladora con la amplitud
seleccionada

       signal start_spi: std_logic; --Para iniciar maestro SPI
       signal tx_data_signal: std_logic_vector (31 downto 0); --Salida de maestro SPI
       signal spi_sck_signal: std_logic; --Conectada a salida "spi_sck"
       signal ss_n_signal: std_logic; --Conectada a salida "dac_cs"
       signal spi_out_signal: std_logic; --Conectada a salida "spi_out"
```

```
       signal saved_key: std_logic_vector(7 downto 0); --Almacena tecla presionada (nota o
flecha)
       signal saved_key2: std_logic_vector(7 downto 0); --Almacena tecla presionada (flecha
mientras se reproduce nota)

       signal carrier_key_store: integer; --Para almacenar el valor actual
       signal wave_enable_store: std_logic; --Para almacenar el valor actual
       signal carrier_octave_store: integer; --Para almacenar el valor actual
       signal saved_key_store: std_logic_vector(7 downto 0); --Para almacenar el valor
actual
       signal saved_key2_store: std_logic_vector(7 downto 0); --Para almacenar el valor
actual

       type eg_state_type is (sStart,                  --estados
                                                         sIdle,
                                                         sTransfer1,
                                                         sRead1,
                                                         sWait1,
                                                         sWait2,
                                                         sNotePressed,
                                                         sTransfer2,
                                                         sRead2,
                                                         sWait3,
                                                         sTransfer3,
                                                         sRead3,
                                                         sEndWithZero,
                                                         sWait4,
                                                         sTransfer4,
                                                         sRead4,
                                                         sFuncPressed,
                                                         sTransfer5,
                                                         sRead5,
                                                         sWait5,
                                                         sTransfer6,
                                                         sRead6,
                                                         sWait6,
                                                         sTransfer7,
                                                         sRead7,
                                                         sWait7,
                                                         sTransfer8,
                                                         sRead8,
                                                         sFuncPressed2,
                                                         sTransfer9,
                                                         sRead9,
                                                         sWait8,
                                                         sTransfer10,
                                                         sRead10,
                                                         sWait9,
                                                         sTransfer11,
                                                         sRead11,
                                                         sWait10,
                                                         sTransfer12,
                                                         sRead12
                                                         );
       signal state_reg, state_next: eg_state_type; --Señales para los estados


       COMPONENT wave_gen
  PORT (
    clk : IN STD_LOGIC;
    ce : IN STD_LOGIC;
    pinc_in : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
    poff_in : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
    sine : OUT STD_LOGIC_VECTOR(11 DOWNTO 0)
  );
END COMPONENT;
```

```
begin

	ps2_receive: entity work.ps2_keyboard
		port map(
			clk => clk,
			ps2_clk => ps2_ck,
			ps2_data => ps2_data_signal,
			ps2_code_new => code_new,
			ps2_code => ps2_code
		);

	carrier_wave : wave_gen
  PORT MAP (
    clk => clk,
    ce => wave_enable,
    pinc_in => carrier_freq,
    poff_in => mod_scaled,
    sine => inv_carrier_out
  );

  mod_wave : wave_gen
  PORT MAP (
    clk => clk,
    ce => wave_enable,
    pinc_in => mod_freq,
    poff_in => "00000000000000000000000000000000",
    sine => inv_mod_out
  );

	spi_sender: entity work.spi_master
		generic map(
			slaves => 1,
			d_width => 32)
		port map(
			clock => clk,
			reset_n => NOT reset,
			enable => start_spi,
			cpol => '0',
			cpha => '0',
			cont => '1',
			clk_div => 0,
			addr => 0,
			tx_data => tx_data_signal,
			sclk => spi_sck_signal,
			busy => ss_n_signal,
			mosi => spi_out_signal);

	carrier_out <= NOT inv_carrier_out(11) & inv_carrier_out (10 downto 0); --Corrige
semiciclos

	tx_data_signal <= "11111111" & "0011" & "0000" & carrier_out & "1110";
	--                                        No importa    cmd      addr       onda
No importa

	disable_others <= "1111";
	dac_clr <= NOT reset;

	spi_sck <= spi_sck_signal;
	dac_cs <= NOT ss_n_signal;
	spi_out <= spi_out_signal;

	ps2_data_signal <= ps2_data;
	ps2_data_out <= ps2_data_signal;

	start_spi <= '1';
```

```
	with carrier_key select

		carrier_note <= 1405 when 1, --Presiona nota 1, da frecuencia para Do0
                                                1488 when 2, --Presiona nota 2, da
frecuencia para Do#0
                                                1577 when 3, --Presiona nota 3, da
frecuencia para Re0
                                                1670 when 4,
                                                1770 when 5,
                                                1875 when 6,
                                                1986 when 7,
                                                2105 when 8,
                                                2230 when 9,
                                                2362 when 10,
                                                2503 when 11,
                                                2652 when 12,
                                                2809 when 13, --Presiona nota 1, da
frecuencia para Do1
                                                2976 when 14,
                                                3153 when 15,
                                                3341 when 16,
                                                3539 when 17,
                                                3750 when others; --18

	--Valores de onda portadora son incrementos de fase calculados con:
	--(frecuencia_musical*(2^phase_width))/reloj_sistema

	--phase width: 32
	--reloj sistema: 50 MHz
	--Frecuencia para Do0: 16.3516 Hz
	--Frecuencia para Do#0: 17.3238 Hz
	--...



	with mod_rate select

		mod_freq <= "0000000" & carrier_freq(31 downto 7)                       when
1, --portadora /128
                                            "00000" & carrier_freq(31 downto 5)
	    when 2, --portadora /32
                                             "0000" & carrier_freq(31 downto 4)
	    when 3, --portadora /16
                                              "000" & carrier_freq(31 downto 3)
	    when 4, --portadora /8
                                               "00" & carrier_freq(31 downto 2)
	    when 5, --portadora /4
                                                '0' & carrier_freq(31 downto 1)
	    when 6, --portadora /2
                                                                    carrier_freq
                                when 7, -- = portadora
                                                                        carrier_freq(30 downto 0)
& '0'	    when others; --portadora x2


	with carrier_octave select --Cada desplazamiento a la izquierda es mult. por 2 y por
ende incremento de octava

		carrier_freq <= "00000000000000000000" &
std_logic_vector(to_unsigned(carrier_note,12))      when 0, --Octava 0
                                                "00000000000000000000" &
std_logic_vector(to_unsigned(carrier_note,12)) & '0' when 1, --Octava 1
                                                "000000000000000000" &
std_logic_vector(to_unsigned(carrier_note,12)) & "00" when 2,
                                                "00000000000000000" &
std_logic_vector(to_unsigned(carrier_note,12)) & "000" when 3,
```

```
                                                    "0000000000000000" &
std_logic_vector(to_unsigned(carrier_note,12)) & "0000" when 4,
                                                    "000000000000000" &
std_logic_vector(to_unsigned(carrier_note,12)) & "00000" when 5,
                                                    "00000000000000" &
std_logic_vector(to_unsigned(carrier_note,12)) & "000000" when 6,
                                                    "0000000000000" &
std_logic_vector(to_unsigned(carrier_note,12)) & "0000000" when others; --7


       with mod_amp select --Amplitud de moduladora, semiciclos correjgidos. Cada
desplazamiento a la derecha es amp/2

              mod_scaled <=                NOT inv_mod_out(11) & inv_mod_out(10 downto 0) &
"0000000000000000000" when 4,
                                            '0' & NOT inv_mod_out(11) & inv_mod_out(10
downto 0) & "00000000000000000000" when 3,
                                            "00" & NOT inv_mod_out(11) & inv_mod_out(10
downto 0) & "00000000000000000000" when 2,
                                            "000" &                 "000000000000"
& "00000000000000000" when others; --1



       carrier_octave_led <= std_logic_vector(to_unsigned(carrier_octave,3));

       mod_rate_led <= std_logic_vector(to_unsigned(mod_rate-1,3));

       mod_amp_led <= std_logic_vector(to_unsigned(mod_amp-1,2));


-- Registro de estados
       process(clk,reset)
       begin
              if (reset='1') then
                     state_reg <= sStart;
              elsif (clk'event and clk='1') then
                     state_reg <= state_next;
                     mod_rate <= mod_rate_next;
                     mod_amp <= mod_amp_next; --Actualiza siguientes valores

                     carrier_key_store <= carrier_key;
                     carrier_octave_store <= carrier_octave;
                     wave_enable_store <= wave_enable;
                     saved_key_store <= saved_key;
                     saved_key2_store <= saved_key2; --Almacena valores

              end if;
       end process;

--Logica de siguiente estado y salidas

       process(state_reg,
                      mod_rate,
                      mod_amp,
                      code_new,
                      ps2_code,
                      carrier_out)
       begin
              state_next <= state_reg;
              mod_rate_next <= mod_rate;
              mod_amp_next <= mod_amp; --Predeterminado si no hay valores siguientes

              carrier_key <= carrier_key_store;
              carrier_octave <= carrier_octave_store;
              wave_enable <= wave_enable_store;
              saved_key <= saved_key_store;
```

```
		saved_key2 <= saved_key2_store; --Almacena valores

		case state_reg is

			when sStart => --Inicializa valores
				carrier_key <= 10;
				carrier_octave <= 4;
				mod_rate_next <= 7;
				mod_amp_next <= 4;
				saved_key <= "00000000";
				saved_key2 <= "00000000";
				state_next <= sIdle;

			when sIdle =>
				if (code_new = '0') then
					state_next <= sTransfer1;
				end if;

			when sTransfer1 => --código PS/2 se está transfiriendo
				if (code_new = '1') then
					state_next <= sRead1;
				end if;

			when sRead1 => --Lee código PS/2
                                                        --Si es tecla musical, establecer
frec, habilitar ondas, ir a NotePressed
                                                  --Si tecla de octava, establecer octava,
regresar a Idle
                                                        --Si se soltó una tecla, ir a Wait1
(se ignorará el siguiente estado, luego se irá a Idle)
                                                        --Si es flecha, ir a Wait4 (se
esperará para el sig. código)
				case ps2_code is
					when "00011100" => --1C, Do
						carrier_key <= 1;
						saved_key <= ps2_code;
						state_next <= sNotePressed;

					when "00011101" => --1D, Do#
						carrier_key <= 2;
						saved_key <= ps2_code;
						state_next <= sNotePressed;


					when "00011011" => --1B, Re
						carrier_key <= 3;
						saved_key <= ps2_code;
						state_next <= sNotePressed;

					when "00100100" => --24, Mib
						carrier_key <= 4;
						saved_key <= ps2_code;
						state_next <= sNotePressed;


					when "00100011" => --23, Mi
						carrier_key <= 5;
						saved_key <= ps2_code;
						state_next <= sNotePressed;


					when "00101011" => --2B, Fa
						carrier_key <= 6;
						saved_key <= ps2_code;
						state_next <= sNotePressed;
```

```
when "00101100" => --2C, Fa#
       carrier_key <= 7;
       saved_key <= ps2_code;
       state_next <= sNotePressed;


when "00110100" => --34, Sol
       carrier_key <= 8;
       saved_key <= ps2_code;
       state_next <= sNotePressed;


when "00110101" => --35, Lab
       carrier_key <= 9;
       saved_key <= ps2_code;
       state_next <= sNotePressed;


when "00110011" => --33, La
       carrier_key <= 10;
       saved_key <= ps2_code;
       state_next <= sNotePressed;


when "00111100" => --3C, Sib
       carrier_key <= 11;
       saved_key <= ps2_code;
       state_next <= sNotePressed;


when "00111011" => --3B, Si
       carrier_key <= 12;
       saved_key <= ps2_code;
       state_next <= sNotePressed;



when "01000010" => --42, Do'
       carrier_key <= 13;
       saved_key <= ps2_code;
       state_next <= sNotePressed;


when "01000100" => --44, Do#'
       carrier_key <= 14;
       saved_key <= ps2_code;
       state_next <= sNotePressed;


when "01001011" => --4B, Re'
       carrier_key <= 15;
       saved_key <= ps2_code;
       state_next <= sNotePressed;


when "01001101" => --4D, Mib'
       carrier_key <= 16;
       saved_key <= ps2_code;
       state_next <= sNotePressed;


when "01001100" => --4C, Mi'
       carrier_key <= 17;
       saved_key <= ps2_code;
       state_next <= sNotePressed;
```

```
                when "01010010" => --52, Fa'
                       carrier_key <= 18;
                       saved_key <= ps2_code;
                       state_next <= sNotePressed;


                when "01000101" => --45, 0
                       carrier_octave <= 0;
                       state_next <= sIdle;


                when "00010110" => --16, 1
                       carrier_octave <= 1;
                       state_next <= sIdle;


                when "00011110" => --1E, 2
                       carrier_octave <= 2;
                       state_next <= sIdle;


                when "00100110" => --26, 3
                       carrier_octave <= 3;
                       state_next <= sIdle;


                when "00100101" => --25, 4
                       carrier_octave <= 4;
                       state_next <= sIdle;


                when "00101110" => --2E, 5
                       carrier_octave <= 5;
                       state_next <= sIdle;


                when "00110110" => --36, 6
                       carrier_octave <= 6;
                       state_next <= sIdle;


                when "00111101" => --3D, 7
                       carrier_octave <= 7;
                       state_next <= sIdle;


                when "11110000" => --F0, break
                       state_next <= sWait1;

                when "11100000" => --E0, flecha
                       state_next <= sWait4;

                when others =>
                       state_next <= sIdle;

        end case;


when sWait1 => --Esperar a que inicie transmisión
        if (code_new = '0') then
                state_next <= sWait2;
        end if;

when sWait2 => --Espera a terminar transmisión, ignorar
        if (code_new = '1') then
                state_next <= sIdle;
```

```
                        end if;

                    when sNotePressed => --Habilitar osciladores, esperar a nueva tecla
                        wave_enable <= '1';
                        if (code_new = '0') then
                            state_next <= sTransfer2;
                        end if;

                    when sTransfer2 => --Esperar a terminar transmisión
                        if (code_new = '1') then
                            state_next <= sRead2;
                        end if;

                    when sRead2 => --Si es nota, cambiar frecuencia y regresar a
NotePressed
                                                    --Si es octava, cambiar octava y
regresar a NotePressed
                                                    --Si es tecla de función, ir a Wait7
                                                    --Si se soltó tecla, ir a Wait3
                        case ps2_code is

                            when "00011100" => --1C, Do
                                carrier_key <= 1;
                                saved_key <= ps2_code;
                                state_next <= sNotePressed;

                            when "00011101" => --1D, Do#
                                carrier_key <= 2;
                                saved_key <= ps2_code;
                                state_next <= sNotePressed;


                            when "00011011" => --1B, Re
                                carrier_key <= 3;
                                saved_key <= ps2_code;
                                state_next <= sNotePressed;

                            when "00100100" => --24, Mib
                                carrier_key <= 4;
                                saved_key <= ps2_code;
                                state_next <= sNotePressed;


                            when "00100011" => --23, Mi
                                carrier_key <= 5;
                                saved_key <= ps2_code;
                                state_next <= sNotePressed;


                            when "00101011" => --2B, Fa
                                carrier_key <= 6;
                                saved_key <= ps2_code;
                                state_next <= sNotePressed;


                            when "00101100" => --2C, Fa#
                                carrier_key <= 7;
                                saved_key <= ps2_code;
                                state_next <= sNotePressed;


                            when "00110100" => --34, Sol
                                carrier_key <= 8;
                                saved_key <= ps2_code;
                                state_next <= sNotePressed;
```

```
when "00110101" => --35, Lab
       carrier_key <= 9;
       saved_key <= ps2_code;
       state_next <= sNotePressed;


when "00110011" => --33, La
       carrier_key <= 10;
       saved_key <= ps2_code;
       state_next <= sNotePressed;


when "00111100" => --3C, Sib
       carrier_key <= 11;
       saved_key <= ps2_code;
       state_next <= sNotePressed;


when "00111011" => --3B, Si
       carrier_key <= 12;
       saved_key <= ps2_code;
       state_next <= sNotePressed;



when "01000010" => --42, Do'
       carrier_key <= 13;
       saved_key <= ps2_code;
       state_next <= sNotePressed;


when "01000100" => --44, Do#'
       carrier_key <= 14;
       saved_key <= ps2_code;
       state_next <= sNotePressed;


when "01001011" => --4B, Re'
       carrier_key <= 15;
       saved_key <= ps2_code;
       state_next <= sNotePressed;


when "01001101" => --4D, Mib'
       carrier_key <= 16;
       saved_key <= ps2_code;
       state_next <= sNotePressed;


when "01001100" => --4C, Mi'
       carrier_key <= 17;
       saved_key <= ps2_code;
       state_next <= sNotePressed;


when "01010010" => --52, Fa'
       carrier_key <= 18;
       saved_key <= ps2_code;
       state_next <= sNotePressed;


when "01000101" => --45, 0
       carrier_octave <= 0;
       state_next <= sNotePressed;


when "00010110" => --16, 1
```

```
                                carrier_octave <= 1;
                                state_next <= sNotePressed;


                        when "00011110" => --1E, 2
                                carrier_octave <= 2;
                                state_next <= sNotePressed;


                        when "00100110" => --26, 3
                                carrier_octave <= 3;
                                state_next <= sNotePressed;


                        when "00100101" => --25, 4
                                carrier_octave <= 4;
                                state_next <= sNotePressed;


                        when "00101110" => --2E, 5
                                carrier_octave <= 5;
                                state_next <= sNotePressed;


                        when "00110110" => --36, 6
                                carrier_octave <= 6;
                                state_next <= sNotePressed;


                        when "00111101" => --3D, 7
                                carrier_octave <= 7;
                                state_next <= sNotePressed;

                        when "11110000" => --F0, break
                                state_next <= sWait3;

                        when "11100000" => --E0, flecha
                                state_next <= sWait7;

                        when others =>
                                state_next <= sNotePressed;

                end case;


        when sWait3 => --Esperar a que inicie transferencia PS/2
                if (code_new = '0') then
                        state_next <= sTransfer3;
                end if;

        when sTransfer3 => --Esperar a que termine transferencia PS/2
                if (code_new = '1') then
                        state_next <= sRead3;
                end if;

        when sRead3 => --Si tecla soltada fué nota actual, ir a EndWithZero
                                                        --            (se deshabilita el
osclador sólo si la tecla soltada fué la ULTIMA tecla musical)
                if (ps2_code = saved_key) then
                        state_next <= sEndWithZero;
                else
                        state_next <= sNotePressed;
                end if;

        when sEndWithZero => --Esperar a que onda pase cerca de cruce por
cero, luego deshabilitar
```

```
                        if ((carrier_out > "011110011010") AND (carrier_out <
"100001100110")) then
                                wave_enable <= '0';
                                state_next <= sIdle;
                        end if;

                when sWait4 =>
                        if (code_new = '0') then
                                state_next <= sTransfer4;
                        end if;


                when sTransfer4 =>
                        if (code_new = '1') then
                                state_next <= sRead4;
                        end if;


                when sRead4 => --Si se presionó flecha, hacer acción.
                        case ps2_code is

                                when "01110100" => --74, Flecha derecha, subir frecuencia
moduladora
                                        if (NOT (mod_rate = 8)) then
                                                mod_rate_next <= mod_rate + 1;
                                        end if;
                                        saved_key <= ps2_code;
                                        state_next <= sFuncPressed;


                                when "01101011" => --6B, Flecha izquierda, bajar
frecuenia moduladora
                                        if (NOT (mod_rate = 1)) then
                                                mod_rate_next <= mod_rate - 1;
                                        end if;
                                        saved_key <= ps2_code;
                                        state_next <= sFuncPressed;


                                when "01110101" => --75, Flecha arriba, subir amplitud
moduladora
                                                if (NOT (mod_amp = 4)) then
                                                mod_amp_next <= mod_amp + 1;
                                        end if;
                                        saved_key <= ps2_code;
                                        state_next <= sFuncPressed;


                                when "01110010" => --72, Flecha abajo, bajar amplitud
moduladora
                                        if (NOT (mod_amp = 1)) then
                                                mod_amp_next <= mod_amp - 1;
                                        end if;
                                        saved_key <= ps2_code;
                                        state_next <= sFuncPressed;


                                when others => --else, ir a Wait1 y esperar sig. código
                                        state_next <= sWait1;

                        end case;


                when sFuncPressed =>
                        if (code_new = '0') then
                                state_next <= sTransfer5;
                        end if;
```

```
                        when sTransfer5 =>
                                if (code_new = '1') then
                                        state_next <= sRead5;
                                end if;

                        when sRead5 =>
                                if (ps2_code = "11100000") then -- E0
                                        state_next <= sWait5;
                                else
                                        state_next <= sFuncPressed;
                                end if;

                        when sWait5 =>
                                if (code_new = '0') then
                                        state_next <= sTransfer6;
                                end if;

                        when sTransfer6 =>
                                if (code_new = '1') then
                                        state_next <= sRead6;
                                end if;

                        when sRead6 =>
                                if (ps2_code = "11110000") then --F0
                                        state_next <= sWait6;
                                else
                                        state_next <= sFuncPressed;
                                end if;


                        when sWait6 =>
                                if (code_new = '0') then
                                        state_next <= sTransfer7;
                                end if;


                        when sTransfer7 =>
                                if (code_new = '1') then
                                        state_next <= sRead7;
                                end if;


                        when sRead7 => --Tecla soltada fué tecla actual de función? Regresar a
Idle
                                                        --else regresa a FuncPressed
                                                        --Esto asegura que FSM regrese a
idle SOLO si se soltó PRIMERA tecla de función
                                                        --  en caso de que se haya
presionado varias simultáneamente
                                if (ps2_code = saved_key) then
                                        state_next <= sIdle;
                                else
                                        state_next <= sFuncPressed;
                                end if;


                        when sWait7 =>
                                if (code_new = '0') then
                                        state_next <= sTransfer8;
                                end if;


                        when sTransfer8 =>
                                if (code_new = '1') then
                                        state_next <= sRead8;
                                end if;
```

```
                when sRead8 => --Se presionó tecla de función MIENTRAS se reproducía
una nota?
                        case ps2_code is

                                when "01110100" => --74, Flecha derecha
                                        if (NOT (mod_rate = 8)) then
                                                mod_rate_next <= mod_rate + 1;
                                        end if;
                                        saved_key2 <= ps2_code;
                                        state_next <= sFuncPressed2;


                                when "01101011" => --6B, Flecha izquierda
                                        if (NOT (mod_rate = 1)) then
                                                mod_rate_next <= mod_rate - 1;
                                        end if;
                                        saved_key2 <= ps2_code;
                                        state_next <= sFuncPressed2;


                                when "01110101" => --75, Flecha arriba
                                                if (NOT (mod_amp = 4)) then
                                                mod_amp_next <= mod_amp + 1;
                                        end if;
                                        saved_key2 <= ps2_code;
                                        state_next <= sFuncPressed2;


                                when "01110010" => --72, Flecha abajo
                                        if (NOT (mod_amp = 1)) then
                                                mod_amp_next <= mod_amp - 1;
                                        end if;
                                        saved_key2 <= ps2_code;
                                        state_next <= sFuncPressed2;


                                when others =>
                                        state_next <= sNotePressed;

                        end case;


                when sFuncPressed2 => --
                        if (code_new = '0') then
                                state_next <= sTransfer9;
                        end if;


                when sTransfer9 =>
                        if (code_new = '1') then
                                state_next <= sRead9;
                        end if;


                when sRead9 =>
                        if (ps2_code = "11100000") then --E0, function
                                state_next <= sWait8;
                        elsif (ps2_code = "11110000") then --F0, break
                                state_next <= sWait10;
                        else
                                state_next <= sFuncPressed2;
                        end if;


                when sWait8 =>
```

```
                        if (code_new = '0') then
                                state_next <= sTransfer10;
                        end if;


                when sTransfer10 =>
                        if (code_new = '1') then
                                state_next <= sRead10;
                        end if;


                when sRead10 =>
                        if (ps2_code = "11110000") then --F0, break
                                state_next <= sWait9;
                        else
                                state_next <= sFuncPressed2;
                        end if;


                when sWait9 =>
                        if (code_new = '0') then
                                state_next <= sTransfer11;
                        end if;


                when sTransfer11 =>
                        if (code_new = '1') then
                                state_next <= sRead11;
                        end if;


                when sRead11 =>
                        if (ps2_code = saved_key2) then
                                state_next <= sNotePressed;
                        else
                                state_next <= sFuncPressed2;
                        end if;


                when sWait10 =>
                        if (code_new = '0') then
                                state_next <= sTransfer12;
                        end if;


                when sTransfer12 =>
                        if (code_new = '1') then
                                state_next <= sRead12;
                        end if;


                when others =>
                        if (ps2_code = saved_key) then
                                state_next <= sEndWithZero;
                        else
                                state_next <= sFuncPressed2;
                        end if;

        end case;

    end process;

end Behavioral;
```

D.2 Módulo anti-rebote

Adquirido de https://eewiki.net/pages/viewpage.action?pageId=28278929.

```
--------------------------------------------------------------------------------
--
--   FileName:         debounce.vhd
--   Dependencies:     none
--   Design Software:  Quartus II 32-bit Version 11.1 Build 173 SJ Full Version
--
--   HDL CODE IS PROVIDED "AS IS."  DIGI-KEY EXPRESSLY DISCLAIMS ANY
--   WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT
--   LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
--   PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL DIGI-KEY
--   BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL
--   DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF
--   PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS
--   BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF),
--   ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.
--
--   Version History
--   Version 1.0 3/26/2012 Scott Larson
--     Initial Public Release
--
--------------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY debounce IS
  GENERIC(
    counter_size  :  INTEGER := 19); --counter size (19 bits gives 10.5ms with 50MHz clock)
  PORT(
    clk     : IN  STD_LOGIC;  --input clock
    button  : IN  STD_LOGIC;  --input signal to be debounced
    result  : OUT STD_LOGIC); --debounced signal
END debounce;

ARCHITECTURE logic OF debounce IS
  SIGNAL flipflops   : STD_LOGIC_VECTOR(1 DOWNTO 0); --input flip flops
  SIGNAL counter_set : STD_LOGIC;                    --sync reset to zero
  SIGNAL counter_out : STD_LOGIC_VECTOR(counter_size DOWNTO 0) := (OTHERS => '0'); --
counter output
BEGIN

  counter_set <= flipflops(0) xor flipflops(1);   --determine when to start/reset counter

  PROCESS(clk)
  BEGIN
    IF(clk'EVENT and clk = '1') THEN
      flipflops(0) <= button;
      flipflops(1) <= flipflops(0);
      If(counter_set = '1') THEN                  --reset counter because input is changing
        counter_out <= (OTHERS => '0');
      ELSIF(counter_out(counter_size) = '0') THEN --stable input time is not yet met
        counter_out <= counter_out + 1;
      ELSE                                        --stable input time is met
        result <= flipflops(1);
      END IF;
    END IF;
  END PROCESS;
END logic;
```

D.3 Receptor de datos de teclado por protocolo serial PS/2

Adquirido de https://eewiki.net/pages/viewpage.action?pageId=28278929.

```
--------------------------------------------------------------------------------
--
--   FileName:         ps2_keyboard.vhd
--   Dependencies:     debounce.vhd
--   Design Software:  Quartus II 32-bit Version 12.1 Build 177 SJ Full Version
--
--   HDL CODE IS PROVIDED "AS IS."  DIGI-KEY EXPRESSLY DISCLAIMS ANY
--   WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT
--   LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
--   PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL DIGI-KEY
--   BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL
--   DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF
--   PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS
--   BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF),
--   ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.
--
--   Version History
--   Version 1.0 11/25/2013 Scott Larson
--     Initial Public Release
--
--------------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY ps2_keyboard IS
  GENERIC(
    clk_freq              : INTEGER := 50_000_000; --system clock frequency in Hz
    debounce_counter_size : INTEGER := 8);         --set such that (2^size)/clk_freq = 5us
(size = 8 for 50MHz)
  PORT(
    clk          : IN  STD_LOGIC;                     --system clock
    ps2_clk      : IN  STD_LOGIC;                     --clock signal from PS/2 keyboard
    ps2_data     : IN  STD_LOGIC;                     --data signal from PS/2 keyboard
    ps2_code_new : OUT STD_LOGIC;                     --flag that new PS/2 code is
available on ps2_code bus
    ps2_code     : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)); --code received from PS/2
END ps2_keyboard;

ARCHITECTURE logic OF ps2_keyboard IS
  SIGNAL sync_ffs     : STD_LOGIC_VECTOR(1 DOWNTO 0);       --synchronizer flip-flops for
PS/2 signals
  SIGNAL ps2_clk_int  : STD_LOGIC;                          --debounced clock signal from
PS/2 keyboard
  SIGNAL ps2_data_int : STD_LOGIC;                          --debounced data signal from
PS/2 keyboard
  SIGNAL ps2_word     : STD_LOGIC_VECTOR(10 DOWNTO 0);      --stores the ps2 data word
  SIGNAL error        : STD_LOGIC;                          --validate parity, start, and
stop bits
  SIGNAL count_idle   : INTEGER RANGE 0 TO clk_freq/18_000; --counter to determine PS/2 is
idle

  --declare debounce component for debouncing PS2 input signals
  COMPONENT debounce IS
    GENERIC(
      counter_size : INTEGER); --debounce period (in seconds) = 2^counter_size/(clk freq in
Hz)
    PORT(
      clk    : IN  STD_LOGIC;  --input clock
```

```
      button : IN  STD_LOGIC;  --input signal to be debounced
      result : OUT STD_LOGIC); --debounced signal
  END COMPONENT;
BEGIN

  --synchronizer flip-flops
  PROCESS(clk)
  BEGIN
    IF(clk'EVENT AND clk = '1') THEN  --rising edge of system clock
      sync_ffs(0) <= ps2_clk;           --synchronize PS/2 clock signal
      sync_ffs(1) <= ps2_data;          --synchronize PS/2 data signal
    END IF;
  END PROCESS;

  --debounce PS2 input signals
  debounce_ps2_clk: debounce
    GENERIC MAP(counter_size => debounce_counter_size)
    PORT MAP(clk => clk, button => sync_ffs(0), result => ps2_clk_int);
  debounce_ps2_data: debounce
    GENERIC MAP(counter_size => debounce_counter_size)
    PORT MAP(clk => clk, button => sync_ffs(1), result => ps2_data_int);

  --input PS2 data
  PROCESS(ps2_clk_int)
  BEGIN
    IF(ps2_clk_int'EVENT AND ps2_clk_int = '0') THEN    --falling edge of PS2 clock
      ps2_word <= ps2_data_int & ps2_word(10 DOWNTO 1);   --shift in PS2 data bit
    END IF;
  END PROCESS;

  --verify that parity, start, and stop bits are all correct
  error <= NOT (NOT ps2_word(0) AND ps2_word(10) AND (ps2_word(9) XOR ps2_word(8) XOR
        ps2_word(7) XOR ps2_word(6) XOR ps2_word(5) XOR ps2_word(4) XOR ps2_word(3) XOR
        ps2_word(2) XOR ps2_word(1)));

  --determine if PS2 port is idle (i.e. last transaction is finished) and output result
  PROCESS(clk)
  BEGIN
    IF(clk'EVENT AND clk = '1') THEN           --rising edge of system clock

      IF(ps2_clk_int = '0') THEN                 --low PS2 clock, PS/2 is active
        count_idle <= 0;                           --reset idle counter
      ELSIF(count_idle /= clk_freq/18_000) THEN  --PS2 clock has been high less than a half
clock period (<55us)
          count_idle <= count_idle + 1;            --continue counting
      END IF;

      IF(count_idle = clk_freq/18_000 AND error = '0') THEN  --idle threshold reached and
no errors detected
        ps2_code_new <= '1';                                   --set flag that new PS/2
code is available
        ps2_code <= ps2_word(8 DOWNTO 1);                      --output new PS/2 code
      ELSE                                                   --PS/2 port active or error
detected
        ps2_code_new <= '0';                                   --set flag that PS/2
transaction is in progress
      END IF;

    END IF;
  END PROCESS;

END logic;
```

D.4 Maestro SPI

Adquirido de https://eewiki.net/pages/viewpage.action?pageId=4096096 y posteriormente modificado ligeramente.

```
---------------------------------------------------------------------------------
--
--   FileName:         spi_master.vhd
--   Dependencies:     none
--   Design Software:  Quartus II Version 9.0 Build 132 SJ Full Version
--
--   HDL CODE IS PROVIDED "AS IS."  DIGI-KEY EXPRESSLY DISCLAIMS ANY
--   WARRANTY OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT
--   LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
--   PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL DIGI-KEY
--   BE LIABLE FOR ANY INCIDENTAL, SPECIAL, INDIRECT OR CONSEQUENTIAL
--   DAMAGES, LOST PROFITS OR LOST DATA, HARM TO YOUR EQUIPMENT, COST OF
--   PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY OR SERVICES, ANY CLAIMS
--   BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF),
--   ANY CLAIMS FOR INDEMNITY OR CONTRIBUTION, OR OTHER SIMILAR COSTS.
--
--   Version History
--   Version 1.0 7/23/2010 Scott Larson
--     Initial Public Release
--   Version 1.1 4/11/2013 Scott Larson
--     Corrected ModelSim simulation error (explicitly reset clk_toggles signal)
--
---------------------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY spi_master IS
  GENERIC(
    slaves  : INTEGER := 4;  --number of spi slaves
    d_width : INTEGER := 2); --data bus width
  PORT(
    clock   : IN     STD_LOGIC;                             --system clock
    reset_n : IN     STD_LOGIC;                             --asynchronous reset
    enable  : IN     STD_LOGIC;                             --initiate transaction
    cpol    : IN     STD_LOGIC;                             --spi clock polarity
    cpha    : IN     STD_LOGIC;                             --spi clock phase
    cont    : IN     STD_LOGIC;                             --continuous mode command
    clk_div : IN     INTEGER;                               --system clock cycles per 1/2
period of sclk
    addr    : IN     INTEGER;                               --address of slave
    tx_data : IN     STD_LOGIC_VECTOR(d_width-1 DOWNTO 0);  --data to transmit
    sclk    : BUFFER STD_LOGIC;                             --spi clock
    mosi    : OUT    STD_LOGIC;                             --master out, slave in
    busy    : OUT    STD_LOGIC);                             --busy / data ready signal
END spi_master;

ARCHITECTURE logic OF spi_master IS
  TYPE machine IS(ready, execute);                          --state machine data type
  SIGNAL state       : machine;                             --current state
  SIGNAL slave       : INTEGER;                             --slave selected for current
transaction
  SIGNAL clk_ratio   : INTEGER;                             --current clk_div
  SIGNAL count       : INTEGER;                             --counter to trigger sclk from
system clock
  SIGNAL clk_toggles : INTEGER RANGE 0 TO d_width*2 + 1;    --count spi clock toggles
```

```
  SIGNAL assert_data : STD_LOGIC;                             --'1' is tx sclk toggle, '0'
is rx sclk toggle
  SIGNAL continue    : STD_LOGIC;                             --flag to continue transaction
  SIGNAL rx_buffer   : STD_LOGIC_VECTOR(d_width-1 DOWNTO 0); --receive data buffer
  SIGNAL tx_buffer   : STD_LOGIC_VECTOR(d_width-1 DOWNTO 0); --transmit data buffer
  SIGNAL last_bit_rx : INTEGER RANGE 0 TO d_width*2;         --last rx data bit location
  SIGNAL ss_n    : STD_LOGIC_VECTOR(slaves-1 DOWNTO 0);   --slave select
  SIGNAL rx_data : STD_LOGIC_VECTOR(d_width-1 DOWNTO 0); --data received
  SIGNAL miso    : STD_LOGIC;                              --master in, slave out
BEGIN
  PROCESS(clock, reset_n)
  BEGIN

    IF(reset_n = '0') THEN        --reset system
      busy <= '1';                --set busy signal
      ss_n <= (OTHERS => '1');    --deassert all slave select lines
      mosi <= 'Z';                --set master out to high impedance
      rx_data <= (OTHERS => '0'); --clear receive data port
      state <= ready;             --go to ready state when reset is exited

    ELSIF(clock'EVENT AND clock = '1') THEN
      CASE state IS               --state machine

        WHEN ready =>
          busy <= '0';             --clock out not busy signal
          ss_n <= (OTHERS => '1'); --set all slave select outputs high
          mosi <= 'Z';             --set mosi output high impedance
          continue <= '0';         --clear continue flag

          --user input to initiate transaction
          IF(enable = '1') THEN
            busy <= '1';             --set busy signal
            IF(addr < slaves) THEN   --check for valid slave address
              slave <= addr;         --clock in current slave selection if valid
            ELSE
              slave <= 0;            --set to first slave if not valid
            END IF;
            IF(clk_div = 0) THEN     --check for valid spi speed
              clk_ratio <= 1;        --set to maximum speed if zero
              count <= 1;            --initiate system-to-spi clock counter
            ELSE
              clk_ratio <= clk_div;  --set to input selection if valid
              count <= clk_div;      --initiate system-to-spi clock counter
            END IF;
            sclk <= cpol;            --set spi clock polarity
            assert_data <= NOT cpha; --set spi clock phase
            tx_buffer <= tx_data;    --clock in data for transmit into buffer
            clk_toggles <= 0;        --initiate clock toggle counter
            last_bit_rx <= d_width*2 + conv_integer(cpha) - 1; --set last rx data bit
            state <= execute;        --proceed to execute state
          ELSE
            state <= ready;          --remain in ready state
          END IF;

        WHEN execute =>
          busy <= '1';        --set busy signal
          ss_n(slave) <= '0'; --set proper slave select output

          --system clock to sclk ratio is met
          IF(count = clk_ratio) THEN
            count <= 1;                     --reset system-to-spi clock counter
            assert_data <= NOT assert_data; --switch transmit/receive indicator
            IF(clk_toggles = d_width*2 + 1) THEN
              clk_toggles <= 0;               --reset spi clock toggles counter
            ELSE
              clk_toggles <= clk_toggles + 1; --increment spi clock toggles counter
            END IF;
```

```
            --spi clock toggle needed
            IF(clk_toggles <= d_width*2 AND ss_n(slave) = '0') THEN
              sclk <= NOT sclk; --toggle spi clock
            END IF;

            --receive spi clock toggle
            IF(assert_data = '0' AND clk_toggles < last_bit_rx + 1 AND ss_n(slave) = '0')
THEN
              rx_buffer <= rx_buffer(d_width-2 DOWNTO 0) & miso; --shift in received bit
            END IF;

            --transmit spi clock toggle
            IF(assert_data = '1' AND clk_toggles < last_bit_rx) THEN
              mosi <= tx_buffer(d_width-1);                    --clock out data bit
              tx_buffer <= tx_buffer(d_width-2 DOWNTO 0) & '0'; --shift data transmit
buffer
            END IF;

            --last data receive, but continue
            IF(clk_toggles = last_bit_rx AND cont = '1') THEN
              tx_buffer <= tx_data;                     --reload transmit buffer
              clk_toggles <= last_bit_rx - d_width*2 + 1; --reset spi clock toggle counter
              continue <= '1';                          --set continue flag
            END IF;

            --normal end of transaction, but continue
            IF(continue = '1') THEN
              continue <= '0';      --clear continue flag
              busy <= '0';          --clock out signal that first receive data is ready
              rx_data <= rx_buffer; --clock out received data to output port
            END IF;

            --end of transaction
            IF((clk_toggles = d_width*2 + 1) AND cont = '0') THEN
              busy <= '0';             --clock out not busy signal
              ss_n <= (OTHERS => '1'); --set all slave selects high
              mosi <= 'Z';             --set mosi output high impedance
              rx_data <= rx_buffer;    --clock out received data to output port
              state <= ready;          --return to ready state
            ELSE                       --not end of transaction
              state <= execute;        --remain in execute state
            END IF;

          ELSE        --system clock to sclk ratio not met
            count <= count + 1; --increment counter
            state <= execute;   --remain in execute state
          END IF;

      END CASE;
    END IF;
  END PROCESS;
END logic;
```