Universitat d'Alacant
Universidad de Alicante
Escola Politècnica Superior
Escuela Politécnica Superior

Proyecto Fin de Carrera
Ingeniería Técnica de Telecomunicación
Sonido e Imagen

Sintetizador FM para uso en tiempo real a partir de Csound

Autor:
Kristian Alonso Stenberg

Tutor:
José Manuel Iñesta Quereda

Septiembre, Año 2013

Agradecimientos

El desarrollo del siguiente proyecto no hubiera sido posible sin el apoyo de determinadas personas que me gustaría nombrar a continuación.

En primer lugar, a mi tutor, José Manuel Iñesta Quereda, por ofrecerme una propuesta que me pareció y me sigue pareciendo interesantísima. Agradecer también todos los consejos otorgados para mejorar el presente proyecto.

En segundo lugar, a mi padre y mi hermano mayor Toñín, por probar el sintetizador y molestarse en venir al ordenador cada vez que conseguía avanzar en el proyecto. A la mejor madre del mundo, que siempre constituirá un ejemplo para mí (Pía). También a mi hermana Kristina, por el apoyo que me ha dado no solo a lo largo de éste proyecto, sino en todo el transcurso de la carrera.

A Mariluz, mi novia y ángel de la guarda. Por todo el amor, cariño y ánimos que ha depositado en mí. Por haberme hecho reír aunque el día fuera gris. Finalmente, a mis suegros (Enrique y Susana) y sus allegados. Por ser mis segundos padres y deleitarme con su compañía y sus detalles.

A mis amigos Henro, Noker, Xuk, Javo, y Moy, por estar toda una vida a mi lado y no haberme fallado jamás; y compañeros de clase, especialmente Gema, Jorge, Lorena y Agustín, con quienes he podido compartir tres maravillosos años.

Muchísimas gracias.

Índice de contenidos

Capítulo 1

Introducción al proyecto

A lo largo de la historia, el ser humano ha tratado de imitar sonidos. Sin ir más lejos, cuando somos pequeños y estamos aprendiendo el idioma materno, intentamos decir las palabras que oímos de nuestros padres. Por otra parte, no sólo hemos imitado, sino creado sonidos nuevos. Las distintas pronunciaciones según el idioma muestran un claro ejemplo de ello.

Parece ser que con la música ha sucedido algo parecido. Hemos procurado copiar sonidos procedentes de otros instrumentos o inventar nuestros propios sonidos, bien sea a través de cambios en parámetros eléctricos (era analógica) o mediante un software específico (era digital).

La herramienta que ha permitido ésto ha sido la *síntesis del sonido*, es decir la creación de sonido a través de sistemas electrónicos sin emplear ninguna fuente sonora. El dispositivo capaz de realizar síntesis de sonido se denomina *sintetizador*. Existen, además, muchas técnicas para realizar síntesis de sonido. Más adelante, profundizaremos en todos éstos conceptos.

El presente proyecto consiste en el diseño de un sintetizador FM (*Frequency Modulation*), es decir, basado en una técnica de síntesis por modulación en frecuencia. Este sintetizador está pensado para trabajar en tiempo real, esto es, que tenemos un tiempo límite para realizar los cálculos de la señal de salida. En este sentido, cuando pulsemos una tecla de nuestro sintetizador, éste debe reaccionar instantáneamente[1].

Este sintetizador se diseñará, por una parte, utilizando el lenguaje de programación *Csound*. Por otra parte, recurriremos a un programa que actúe como una interfaz (o *front-end*) de *Csound* y permita construir los elementos gráficos del sintetizador (*widgets*) con los que el usuario podrá interactuar. El programa que se encargará de éste propósito se denomina *QuteCsound*.

[1] En un tiempo menor a 10 milisegundos (ms).

1.1. Origen del proyecto

El siguiente proyecto nace a raíz de la propuesta de José Manuel Iñesta Quereda, a fin de poder enseñar de un modo más fácil y claro la técnica de síntesis FM al alumnado que estudia la asignatura de *Síntesis Digital de Sonido*.

Para ello, se requiere afrontar el diseño de un sintetizador FM que sea capaz de trabajar en tiempo real, pueda ser controlado vía MIDI[2] y tenga una interfaz gráfica que permita al usuario un control intuitivo y una monitorización de los distintos parámetros.

De este modo, mediante el sintetizador diseñado se desea que, tanto el alumno como aquellas personas interesadas en el mundo de la síntesis de sonido: estudien los parámetros relacionados con la síntesis FM (pudiendo contrastar en todo momento la influencia de los mismos con la práctica), vean qué repercusión tienen éstos en el espectro de la señal de salida y observen cómo se traduce todo ello en cómo oímos dicha señal.

1.2. Objetivos

Los objetivos principales del presente proyecto se detallan a continuación:

- Comprender de un modo sencillo la síntesis por modulación en frecuencia (FM), así como sus ventajas e inconvenientes.
- Entender cómo funciona un sintetizador FM básico.
- Observar qué parámetros intervienen en un sintetizador FM.
- Estudiar el efecto de dichos parámetros en el espectro de la señal de salida.
- Aprender a implementar diseños de sintetizadores FM en *Csound*.
- Familiarizarse con el entorno *QuteCsound* para realizar sintetizadores en tiempo real con interfaz gráfica.
- Crear una herramienta que apoye y facilite la tarea docente de la síntesis FM.
- Promover software libre y gratuito para fines didácticos y/o personales.

[2] *Musical Instrument Digital Interface.*

1.3. Metodología de trabajo

Para conseguir los objetivos descritos en el apartado anterior, resulta imprescindible seguir una metodología de trabajo distribuida en varias fases que nos acerque de manera gradual a nuestro sintetizador.

- Fase I: Establecer los fundamentos teóricos.

  Deberemos comenzar el camino pasando por todos los contenidos teóricos necesarios que nos permitan comprender qué es un sintetizador. Esto incluye desde las diversas técnicas existentes para realizar síntesis de sonido hasta las diferencias entre un sintetizador analógico y uno digital. En este caso, trataremos en mayor profundidad la técnica de síntesis FM, ya que nuestro sintetizador se basará en ésta. Veremos cómo podremos definir un instrumento a partir del sintetizador FM, dependiendo de la configuración de parámetros que escojamos.

  También comprenderemos el funcionamiento del lenguaje *Csound*, así como su estructura y la sintaxis que sigue. *Csound* será el mecanismo mediante el cual podremos crear el sintetizador, haciendo uso de los conocimientos adquiridos hasta entonces.

  Finalmente, estudiaremos, por una parte, cómo se intercambia la información entre *Csound* y un controlador externo, a través del protocolo MIDI; por otra parte, aprenderemos a utilizar el software *QuteCsound*, que será el que nos permitirá el uso del sintetizador en tiempo real y la integración de la interfaz gráfica.

- Fase II: Diseño inicial del sistema.

  Implementaremos una versión inicial del sistema en *Csound*, que tendrá que ser capaz de cubrir las funciones básicas propias de un sintetizador FM:

  - Variación del índice de modulación.
  - Modificación de las frecuencias de las señales portadora y moduladora.
  - Envolventes de amplitud ADSR[3] ajustables para las señales portadora y moduladora.

  Cabe destacar que el sistema inicial deberá ser capaz de trabajar en tiempo real y podrá ser controlado por un teclado externo[4] vía MIDI.

[3] *Attack - Decay - Sustain - Release*. En el siguiente capítulo se indagará en este tipo de envolventes.

[4] En el caso que nos ocupa, nuestro controlador externo será un *Casio Privia PX - 120*.

Por esta razón, *Csound* deberá entender los mensajes que reciba desde dicho teclado.

En este momento, realizaremos una interfaz sencilla empleando el programa *QuteCsound*, para verificar que se cumplen las funciones básicas del sintetizador. Comprobaremos también que la conexión MIDI es correcta y que existe una latencia[5] adecuada (inferior a 10 ms).

- Fase III: Ampliación y optimización del sistema.

  Una vez asegurados de que el funcionamiento del sintetizador FM es correcto, procederemos a mejorar las capacidades de nuestro sistema, añadiendo nuevos módulos que cumplan funciones específicas:

  - Selección de las formas de onda de las señales portadora y moduladora, así como la visualización de las mismas.
  - Posibilidad de representación gráfica del espectro de la señal de salida en tiempo real.
  - Módulo de efectos de retardo (*delay*) y reverberación (*reverb*), con parámetros ajustables y con interruptor de encendido/apagado.
  - Oscilador de baja frecuencia (LFO) para aplicar efecto *trémolo* o *vibrato* a la señal de salida.
  - Potenciómetro para volumen de señal de salida con display que se ilumina si la señal distorsiona.
  - Display ON, que se activa si está encendido el sistema y se desactiva en caso contrario.
  - Display NOTA ON, que se enciende si se recibe la pulsación de una tecla del controlador MIDI.
  - Selector y almacenador de PRESETS[6].
  - Botones de encendido y apagado del sistema.

  Además de éstas nuevas características, será necesario hacer más eficiente el código de nuestro sintetizador, modularizándolo en funciones (denominadas *opcodes* en *Csound*). Por último, optimizaremos la interfaz gráfica, distribuyendo los elementos del sintetizador en un *rack* o rejilla.

[5] Tiempo que transcurre entre la pulsación de una tecla del controlador y la reacción del sintetizador.

[6] Configuración establecida de parámetros del sintetizador para formar un instrumento determinado.

- Fase IV: Validación del sistema final.

  El sistema final se someterá a diversas pruebas concretas para revisar el comportamiento del mismo. Los nuevas características añadidas deben dar respuesta a su función.

  Llegados a este punto, procederemos al diseño de los instrumentos, almacenándolos en *presets*. Con ello, podremos hacernos una idea de la versatilidad sonora que ofrece el sintetizador.

  Anexo a esta memoria, el lector también encontrará diversos vídeos a modo de tutorial, mediante los cuales podrá aprender a utilizar el sintetizador, así como analizar las partes que lo componen.

1.4. Antecedentes

Los antecedentes del presente proyecto pueden dividirse en dos grandes grupos: los sintetizadores físicos (*hardware*) y los sintetizadores virtuales (*software*).

Los sintetizadores también pueden clasificarse en *analógicos* y *digitales*.

En pocas palabras, podríamos decir que los sintetizadores analógicos son aquellos cuya señal sonora de salida es una corriente alterna, es decir una señal *continua* en el tiempo, y que por tanto, toma un valor concreto en cualquier instante de tiempo.

Por otro lado, los sintetizadores digitales presentan como señal sonora de salida un vector (o *array*) de números que se repite periódicamente. Ésta señal, en cambio, se trata de un señal *discreta* en el tiempo, es decir, sólo puede tomar valores concretos en determinados instantes de tiempo (véase Fig. 1.1).

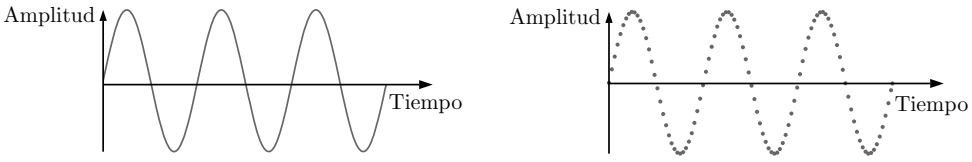


Figura 1.1: Señal continua (izqda) y discreta (dcha).

A continuación, veremos los modelos de sintetizadores FM *hardware*/*software* más representativos hasta la fecha del presente proyecto.

1.4.1. Sintetizadores FM hardware

Los sintetizadores físicos digitales basados en FM saltaron a la escena a partir de 1977[7]. A día de hoy, existen numerosos sintetizadores que aprovechan ésta técnica. A continuación, analizaremos los modelos más representativos que preceden a nuestro proyecto.

1.4.1.1. Yamaha *DX7*

El *Yamaha DX7* ha sido el sintetizador FM por excelencia a lo largo de las últimas décadas. Este revolucionario modelo, fue lanzado por *Yamaha Corporation* en el año 1983. Este teclado programable de 61 teclas sensibles al tacto, disponía de una memoria interna capaz de almacenar 32 voces o instrumentos. No obstante, existía la posibilidad de introducirle un cartucho externo que contenía 64 voces extra, haciendo un total de 96.

Figura 1.2: Sintetizador *Yamaha DX7*. Vista en ángulo y trasera.

El intérprete podía crear nuevas voces desde cero o editar las ya existentes, pudiendo guardarlas bien en la memoria interna del dispositivo, o bien en el cartucho extraíble.

Este sintetizador FM polifónico (16 notas) partía de 6 operadores y 32 algoritmos seleccionables[8] para llevar a cabo la creación de sonidos (Fig. 1.3). A cada uno de éstos operadores se les podía asignar una envolvente independiente con 8 parámetros de ajuste: 4 niveles (*levels*) y 4 tiempos (*rates*).

[7] Comercialización del *Synclavier*, primer sintetizador digital FM.

[8] Los *operadores* son osciladores, mientras que los *algoritmos* son el modo de conectar dichos operadores entre sí.

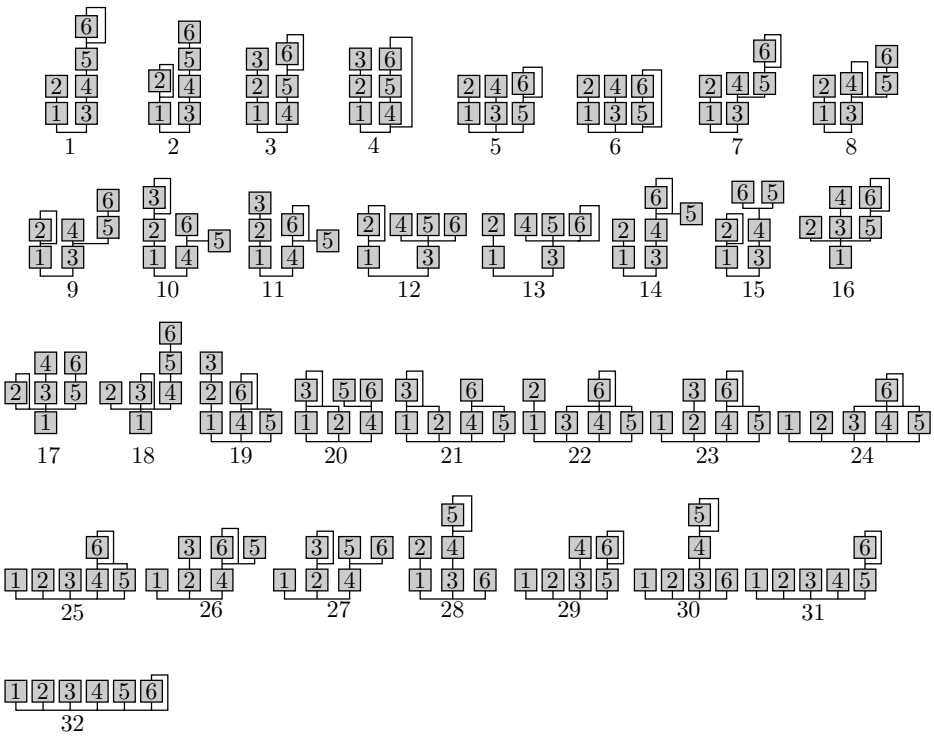


Figura 1.3: Librería de algoritmos del sintetizador *Yamaha DX*7. Cada bloque numerado simboliza un operador y cada configuración un algoritmo.

Las envolventes generadas por éste sintetizador, presentaban mayor flexibilidad frente a las ADSR convencionales, con 4 parámetros ajustables (Fig. 1.4).

El *DX*7 implementaba una de las primeras interfaces MIDI, posibilitando la comunicación de éste con otros dispositivos de la serie *DX*. Con ello, desde nuestro sintetizador lográbamos controlar (por ejemplo) otro al que estuviéramos conectado, pudiendo tocar los dos simultáneamente. Ésto no sólo se limitaba a dos sintetizadores, sino que podíamos controlar más, dependiendo de la configuración MIDI que se realizara (véase Fig. 1.5).

De entre los artistas más famosos que han utilizado el *DX*7, se pueden nombrar *Deep Purple*, *Kraftwerk*, *U*2, *Queen* o *Michael Jackson*.

Si desea escuchar un ejemplo de piano electrónico del *Yamaha DX*7, abra el siguiente archivo de audio, incluido en la carpeta `Audio` anexa a esta memoria:

`1.-Yamaha DX-7-E.Piano.mp3`

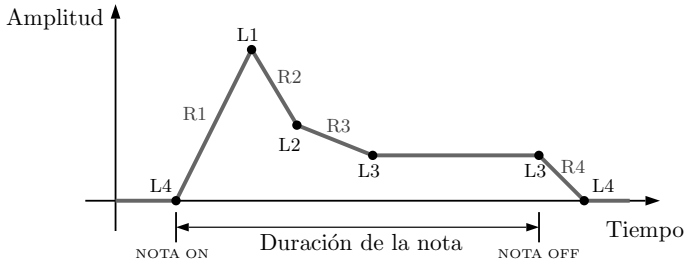


Figura 1.4: Envolvente programable del *Yamaha DX*7. Los parámetros L asignan la amplitud de las distintas fases de la envolvente. Los parámetros R establecen la duración de cada fase.

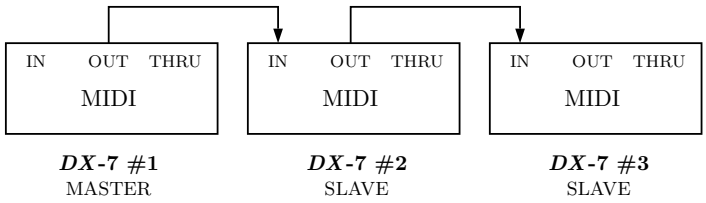


Figura 1.5: Ejemplo de configuración MIDI *Maestro-Esclavo* para tres sintetizadores *Yamaha DX*7. El primer sintetizador actúa como *Maestro* del segundo sintetizador (*Esclavo*), por lo que el *DX*7 #1 puede tanto tocar como programar el *DX*7 #2. No obstante, el primer sintetizador no puede ser *Maestro* del tercero. Sólo podrá ser *Maestro* del tercero el segundo sintetizador.

El principal inconveniente del *DX*7 fue la complejidad que presentaba a la hora de programar los instrumentos a través de la interfaz del dispositivo (poco intuitiva para la mayoría de intérpretes).

Este modelo obtuvo un nuevo cambio de imagen, a partir de 1986, cuando aparecieron tres importantes actualizaciones: el $DX7S$, el $DX7IID$ y el $DX7IIFD$.

- ***DX7S***: La principal mejora que implementaba este sintetizador fue la nueva circuitería de 16 bit, que optimizaba tanto la calidad del sonido como la relación señal a ruido. El nuevo chasis y los pulsadores hacían del $DX7S$ un ejemplar más robusto que su hermano menor $DX7$. Contenía sliders más largos, para facilitar el control en tiempo real; se dobló la memoria y por consiguiente, la cantidad de voces que podía almacenar, haciendo un total de 64. Finalmente, este modelo incluía otras mejoras como una nueva interfaz MIDI, sistema de microafinación o múltiples LFO's, entre otras.

- ***DX7IID***: Con este modelo se ponía de manifiesto la capacidad multitímbrica mediante dos nuevos modos seleccionables: el modo *dual* y el modo *split*. Mediante el modo *dual* se podían seleccionar, de entre todas, dos voces para que sonaran al unísono al pulsar cualquier tecla, consiguiendo así una gran combinación de nuevos sonidos[9]. El modo *split* permitía al usuario "separar" el teclado en dos instrumentos diferentes, de modo que a cada parte del teclado se le podía asignar la voz que se deseara. Esto no era posible realizarlo con los modelos anteriores $DX7$ y $DX7S$[10].

- ***DX7IIFD***: Éste ejemplar incluía una unidad para cartucho de memoria (*Floppy Drive*), mediante el cual se ampliaba la memoria total en 1 MB, pudiendo guardar más de mil voces, almacenar escalado fraccional y otros tipos de datos, como la información procedente de otros dispositivos MIDI.

Figura 1.6: Sintetizador *Yamaha* $DX7S$ (izqda) y $DX7IID$ (dcha).

Figura 1.7: Sintetizador *Yamaha* $DX7IIFD$.

[9]La D (*Dual*) al final del nombre de éste sintetizador responde a la capacidad bitímbrica del mismo.

[10]La S (*Single*) del título se debe a que presentaba capacidad monotímibrica.

1.4.1.2. Yamaha *DX*1

El *Yamaha DX*1 fue el sintetizador más caro[11], grande y sofisticado de la serie *DX*. Fue lanzado en el año 1984 y, como el resto de modelos de la gama, estaba basado en síntesis FM digital (véase Fig. 1.8).

El *DX*1 contaba con un novedoso sistema FM de dos canales independientes (*Channel A* y *Channel B*), de tal modo que cada canal podía utilizar su propio conjunto de 6 operadores, es decir, cada canal se podía comportar como un sintetizador FM. Ésto era equivalente a tener dos *DX*7 unidos en un mismo dispositivo. Por este motivo, disponía del doble de memoria interna que el *DX*7, permitiendo además instalar un máximo de dos cartuchos externos de memoria con capacidad de 64 voces cada uno.

Figura 1.8: Vista frontal del sintetizador *Yamaha DX*1.

Si el intérprete seleccionaba el modo *dual*, asignaba dos voces independientes para que sonaran a la vez cuando se pulsara cualquier tecla, consiguiendo una polifonía de 16 notas por voz. Por otra parte, mediante el modo *single* (una voz) o *split* (dos voces independientes, una para cada región del teclado), se conseguía una polifonía de 32 notas en el primer caso, y 16 notas por voz en el segundo.

La librería de 32 algoritmos que incorporaba era idéntica a la del *DX*7. El teclado de 73 teclas contrapesadas, sensibles tanto a la velocidad como al *aftertouch*, añadían un importante factor de realismo a la interpretación.

Con el *DX*1, *Yamaha* conseguía facilitar a los usuarios la tarea de programación del sintetizador, muy engorrosa hasta entonces en los anteriores modelos de la serie. Ello se logró gracias al nuevo e intuitivo sistema de visualización mediante displays *LED* y *LCD*. Ahora el usuario podía observar y modificar en todo momento el valor de los parámetros de ajuste: amplitud de modulación, envolventes, niveles, velocidad...etc (Fig. 1.9).

[11]El precio del *Yamaha DX*1 durante 1985 fue de 13.900 dólares. Por ello, sólo se fabricaron 140 unidades.

Figura 1.9: Detalle de los displays y nuevos pulsadores del *Yamaha DX*1.

Un posible inconveniente que no se subsanó en este modelo fue el hecho de carecer de efectos internos (*FX*).

Actualmente, el *DX*1 es utilizado por artistas como *Kraftwerk*, *Vince Clarke*, *Elton John* o *Herbie Hancock*.

En el siguiente enlace, podrá ver un vídeo en el que se muestran ejemplos sonoros del *Yamaha DX*1 y se modifican los parámetros del mismo:

`http://youtu.be/bLhLpcXKOgI`

1.4.1.3. Yamaha *FS*1*R*

Tras varios años sin la aparición de nuevos sintetizadores basados en FM, la compañía *Yamaha* esperó hasta 1998 para lanzar al mercado uno de los sistemas más poderosos en lo que a FM se refería: el *FS*1*R* (Fig. 1.10).

El *Yamaha FS*1*R* superaba con creces a sus antecesores (series *DX* y *TX*[12]), ya que implementaba 16 operadores y nada menos que 88 algoritmos. Este sintetizador presentaba dos tecnologías para llevar a cabo la síntesis de sonido: síntesis FM y un nuevo concepto denominado síntesis FS (*Formant Shaping*). La síntesis FS se fundamenta en el control de *formantes*.

Los formantes consisten en regiones del espectro sonoro de la voz en las que la amplitud es máxima. En este sentido, gracias a un conjunto de formantes determinado, podemos ser capaces de distinguir un sonido u otro del habla humana, como por ejemplo una vocal.

[12]La serie *TX* de *Yamaha* consistía en varios sintetizadores FM que versionaban a los de la serie *DX*. Tenían la particularidad de que no disponían de teclado físico propio y, por consiguiente, se tenían que controlar y/o programar bien a través de otro teclado *DX*, o bien a través de un ordenador. Los modelos de ésta serie se solían instalar atornillados en *racks*, salvo el modelo *TX*7.

Para llevar a cabo la síntesis FS, el *FS*1*R* partía de 16 operadores formantes[13] (8 sonoros y 8 sordos). Los operadores *sonoros* se encargaban de crear los sonidos tonales, que podían reproducirse mediante un teclado o un controlador MIDI. Los operadores *sordos* cubrían la función de generar componentes de ruido, similares a las del habla humana cuando se pronuncian, por ejemplo, consonantes fricativas (`f`, `s` o `x`).

Modificando los parámetros de cada operador formante (frecuencia central, amplitud, ancho de banda y falda), se formaban sonidos próximos a los del timbre humano que aún no habían aparecido en otros sintetizadores FM (Fig. 1.11).

El *Yamaha FS*1*R* era un sintetizador que carecía de teclado interno, por lo que necesariamente debía conectarse a un teclado MIDI, un sintetizador o un ordenador con un software de secuenciación (Fig. 1.12).

Figura 1.10: Sintetizador *Yamaha FS*1*R*.

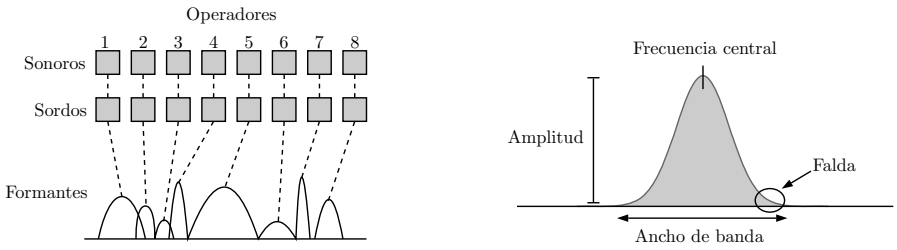
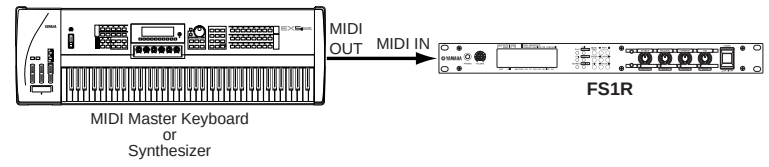
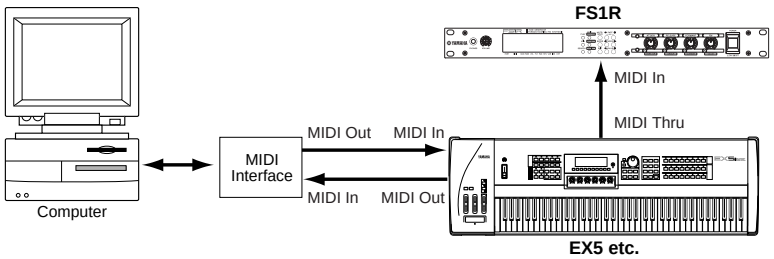


Figura 1.11: (a) Diagrama básico de configuración de operadores para crear formantes en el *Yamaha FS*1*R*. (b) Parámetros modificables de los operadores formantes del *Yamaha FS*1*R*.

[13]En este caso, *Yamaha* continuaba utilizando el término *operadores* debido a que los operadores formantes podían combinarse en distintos algoritmos para crear un sonido determinado, siguiendo la misma pauta que en la serie *DX*.



(a)



(b)

Figura 1.12: Ejemplo de configuraciones MIDI del *Yamaha FS*1*R* (extraído del manual del usuario). (a) Conexión de teclado/sintetizador MIDI a *FS*1*R*. (b) Conexión de ordenador o secuenciador a *FS*1*R*.

Este sintetizador podía reproducir simultáneamente un máximo de 4 timbres (o partes), con una polifonía de 32 notas en caso de no utilizar filtro, o de 16 notas en caso de que el usuario lo activase. Se trataba de filtros dinámicos modelados paso-bajo, paso-alto o paso-banda, con atenuaciones de 12, 18 o 24 dB/octava. A diferencia de los anteriores modelos, el *FS*1*R* poseía dos LFO's, con los que poder realizar modulaciones.

Disponía de una memoria capaz de almacenar 1536 voces, 512 *performances*[14] y 96 secuencias de formantes. Incorporaba efectos internos como reverberación, *delay*, *flanger* o *auto-wah* entre otros, además de inserciones y otro módulo para ecualización.

Una ventaja de éste modelo era la retrocompatibilidad con los sintetizadores de la serie *DX*. Así pues, se podía enviar un ajuste de sonido determinado (o *patch*) vía MIDI desde un *DX*7 y el *FS*1*R* lo convertía en un *patch* idéntico en el nuevo sintetizador.

Como sucedía en otros sintetizadores FM, la desventaja del *FS*1*R* consistía en la complejidad de su interfaz. Con cientos de menús y submenús a los que poder acceder y una pantalla *LCD* de dimensiones muy reducidas, la

[14]El término *performance* se refiere a una combinación de 1 a 4 partes (timbres). A cada parte se le puede asignar una voz y se le pueden modificar varios parámetros que definirán la mezcla final de las partes utilizadas en el *performance*.

programación del sintetizador resultaba un trabajo realmente tedioso. Afortunadamente, más adelante aparecieron softwares multiplataforma para facilitar la tarea de programación del *FS*1*R*[15].

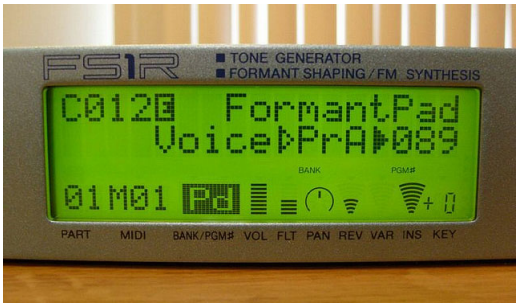


Figura 1.13: Detalle de la pantalla *LCD* del *Yamaha FS*1*R*.

Sin embargo, ésta dificultad a la hora de programar el *FS*1*R* se veía claramente compensada por la gran cantidad de diferentes sonidos que era capaz de generar, más que ningún otro sintetizador FM por el momento.

Tras un pésimo periodo de ventas, el *Yamaha FS*1*R* sólo duró un año en el mercado. Actualmente, es difícil adquirir uno y las pocas existencias que quedan tienen precios elevados.

Squarepusher o *Sin* son algunos ejemplos de grupos musicales que han utilzado el *Yamaha FS*1*R*.

Acceda a la carpeta `Audio` para reproducir un ejemplo de secuencia de varios sonidos del *Yamaha FS*1*R*:

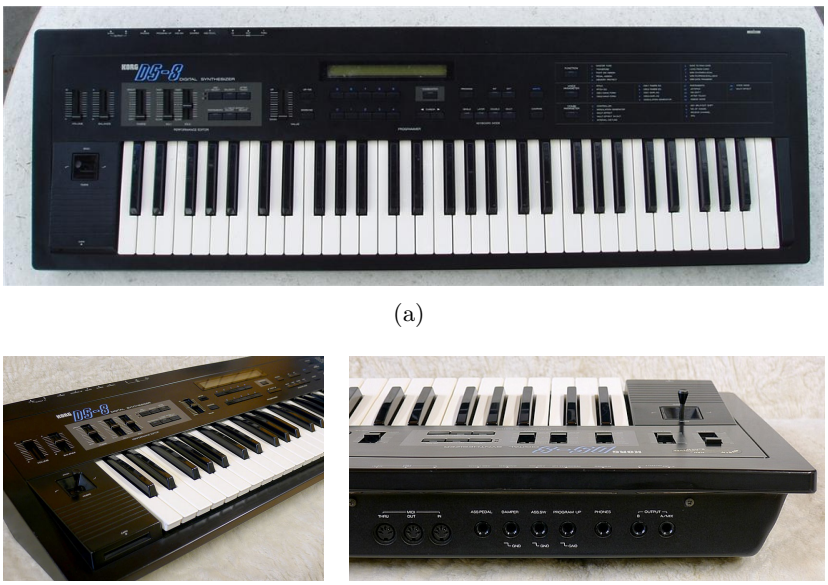
`2.-Yamaha FS1R-Ejemplo.mp3`

1.4.1.4. Korg *DS*-8

La empresa *Korg* también ocupó lugar en la síntesis FM cuando en 1987 comercializó su sintetizador digital: el *DS*-8. Éste modelo competía con algunos de los sintetizadores más importantes del momento: el *Yamaha DX*7 y el *Roland D*-50[16]. Se trata de un sintetizador FM con 4 operadores por voz, polifonía de 8 notas y capacidad multitimbre de 8 voces como máximo.

[15]Un ejemplo de software gratuito es el *FS*1*R* Editor.
Sitio web: `http://synth-voice.sakura.ne.jp/fs1r_editor_english.html`

[16]El *Roland D*-50 fue un sintetizador muy popular en la década de los 80. Estaba basado en síntesis aritmética lineal (*LAS*) y ofrecía sonidos típicos de la síntesis sustractiva analógica. En definitiva, un fuerte contrincante del *DX*7, con la ventaja de ser más fácil de programar.



(a)

(b)

Figura 1.14: Sintetizador *Korg DS*-8. (a) Vista frontal. (b) Vista lateral y trasera.

Aunque disponía de una memoria interna de 100 voces, cabía la posibilidad de añadir más, gracias al cartucho ROM opcional (memoria de lectura). Además de ello, se encontraba en el mercado otro cartucho externo de memoria RAM (memoria de acceso aleatorio), que permitía tanto leer como almacenar las voces programadas por el usuario (Fig. 1.15).

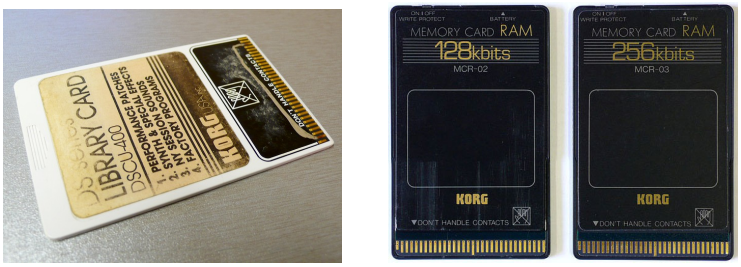


Figura 1.15: Memorias extraíbles del *Korg DS*-8. Memoria ROM *DSCU*-400 (izqda) y dos versiones de memoria RAM: *MCR*-02 y *MCR*-03 (dcha).

Una particularidad del *DS*-8 era que incluía un módulo reservado para efectos (*delay*, *chorus*, *flanger*, etc), así como controles para los moduladores, las envolventes y el LFO. El teclado estaba constituido por 61 teclas sensibles a la velocidad y *aftertouch*.

Otra característica interesante era el *joystick* situado en la parte izquierda del sintetizador. Existían cuatro posibles direcciones a las que desplazar el joystick: arriba, abajo, izquierda y derecha. Si se desplazaba hacia arriba se producía un efecto de *vibrato* al tocar la voz seleccionada, mientras que si se movía hacia abajo se generaba un *trémolo* o un *wah-wah*. Finalmente, si desplazábamos el joystick a la derecha, el *pitch* (o tono) se hacía más agudo, mientras que el timbre adquiría más brillo. Por otro lado, si se movía hacia la izquierda el *pitch* se hacía más grave y el timbre más suave.

Incluía tres *sliders* para modificar el timbre y los dos generadores de envolventes en tiempo real. Acceda al siguiente enlace si desea ver un vídeo demostrativo de los controles mencionados:

`http://youtu.be/7OQ3s8rfHVA`

Por último, abra la carpeta `Audio` y reproduzca el siguiente archivo para escuchar un ejemplo de composición realizada mediante el *Korg DS*-8:

`3.-Korg DS-8-Ejemplo.mp3`

1.4.2. Sintetizadores FM software

Los sintetizadores FM *software* (o virtuales) comenzaron a tener repercusión a finales de los 90 y principios del año 2000. No debemos olvidar que en éste momento se produjeron notables mejoras en la potencia de cálculo de los ordenadores existentes (evolución de los microprocesadores, DSP's[17], memoria RAM, etc), haciendo posible trabajar con el sintetizador en tiempo real y poniéndose a la altura de los sintetizadores *hardware*. En éste apartado, trataremos brevemente los programas principales que se encuentran en auge en el campo de la síntesis FM.

1.4.2.1. Native Instruments *FM7*

En 2002, la compañía *Native Instruments* decide lanzar al mercado un programa multiplataforma que sea capaz de emular los sonidos de los sintetizadores de la serie *DX*: así nace el *FM*7 (Fig. 1.16).

Mediante éste *software* se lograba, por una parte, proporcionar al usuario una interfaz muy intuitiva con la que comprender la síntesis FM de un modo visual y sencillo; por otra parte, hacía posible disponer en un mismo *plugin* todos los sintetizadores de la serie *DX* y reproducirlos fielmente desde nuestro ordenador, pudiendo importar hasta los propios archivos de voces *SysEx* de los sintetizadores originales.

[17] Procesadores Digitales de Señal.

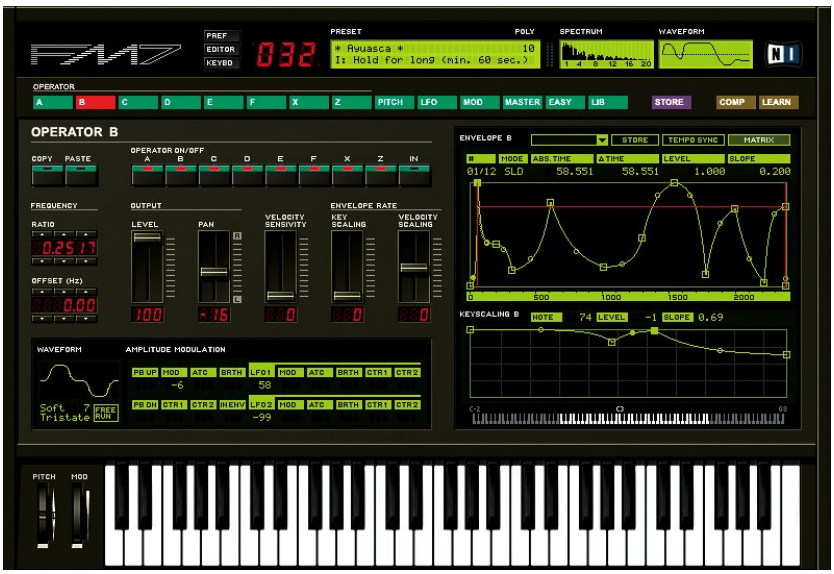


Figura 1.16: Software *Native Instruments FM*7.

El *FM*7 parte de ocho operadores que pueden ser editados bien desde la interfaz gráfica del programa o bien desde un controlador MIDI. Entre dichos operadores podemos distinguir dos grupos: uno de 6 operadores (A-F) que se dedican exclusivamente a la síntesis FM y otro de 2 operadores (X,Z) que amplían la funcionalidad. El primer grupo de operadores, del mismo modo que en la serie *DX*, se encargan de generar una señal (con 32 formas de onda seleccionables) que puede ser utilizada como portadora o moduladora. A su vez, éstos operadores pueden combinarse fácilmente mediante una matriz programable (Fig. 1.17), en la que el usuario puede realizar las conexiones que desee gráficamente (en cascada, paralelo o realmientación). En cambio, el segundo grupo de operadores se centra en añadir ruido y distorsión (operador X) e implementar dos filtros resonadores multimodo (operador Z), superando las capacidades de los sintetizadores *DX* clásicos.

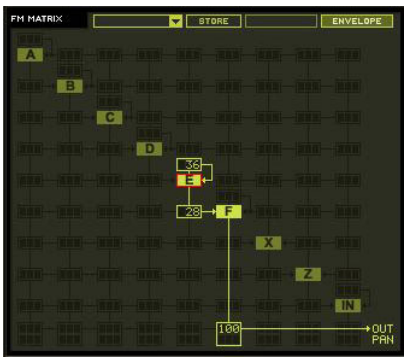


Figura 1.17: Matriz programable del *Native Instruments FM*7.

Éste sintetizador virtual integra inicialmente dos bancos de 128 voces nuevas, un banco de 128 voces originales del *DX*200 y el set completo de voces del *DX*7 con *presets* adicionales. Todas las voces se cargan desde la matriz mencionada.

Las envolventes de cada operador pueden personalizarse gráficamente, permitiendo crear una curva de hasta 30 puntos (superando con creces la del *DX*7, que admitía únicamente 6 puntos). Existen también otros parámetros asociados a los operadores que podemos variar desde la pantalla, como el nivel de amplitud, el rango de frecuencias o incluso la panorámica estéreo (entre muchos otros).

El *software* implementa dos LFO's por cada voz, un teclado virtual de 73 teclas y diversos efectos estéreo semejantes a los mencionados hasta ahora.

Definitivamente, uno de los puntos más destacables del sintetizador residiría en la función MIDI *Learn*. Gracias a ella, el usuario puede asignar mediante un *click* cualquier parámetro del sintetizador *FM*7 al control que desee del teclado MIDI externo (*fader*, *knob*, pulsador...etc), por lo que puede resultar muy interesante para el intérprete a la hora de trabajar tanto a nivel doméstico como en actuaciones.

El *FM*7 se puede utilizar como aplicación independiente, mediante un *driver* de sonido (*ASIO*, *MME*, *Direct X*, etc), o como *plugin* de otros programas (*VST*, *DX*i, *Core Audio*, etc).

Para finalizar, escuche el archivo de audio adjunto. En él podrá oír una instrumental creada exclusivamente con el sintetizador *FM*7:

4.-Native Instruments FM7-Ejemplo.mp3

1.4.2.2. Native Instruments FM8

El *FM*8 constituye el principal sucesor del *FM*7 y fue lanzado en el año 2006. En esta ocasión, el *FM*8 aparece como un potente sintetizador FM que mejora a su hermano menor, no sólo en interfaz sino también en funcionalidad.

Comenzando por los aspectos más básicos, si bien presenta los $6 + 2$ operadores del *FM*7, incluye una importante implementación: un operador de entrada (*IN-operator*). El operador de entrada tiene especial hincapié cuando se utiliza el *FM*8 como *plugin* de efectos, denominándose en este caso *FM*8*fx*.

Figura 1.18: Software *Native Instruments FM*8.

Cualquier señal de audio que le inyectemos al operador de entrada podremos emplearla como salida del mismo, pudiendo modularse o modular otras señales mediante ésta, abriendo un abanico de infinitas posibilidades sonoras.

El *FM*8 presenta una polifonía máxima de 64 notas por instancia[18], una envolvente multisegmento por cada operador, 32 formas de onda seleccionables (incluyendo las de la serie *TX* y algunos formantes), 2 LFO's, y gracias al operador Z, 2 filtros resonadores similares a los del *FM*7, con la diferencia de poder conectarlos en cascada o en paralelo.

Puesto que el *FM*8 es totalmente retrocompatible con el *FM*7, no sólo contiene todos los sonidos del modelo anterior, sino 200 sonidos nuevos que aprovechan las nuevas características del software. El módulo de efectos, ahora se distribuye en forma de *rack* con 12 nuevos efectos, ampliando la colección de su antecesor (Fig. 1.19). Los algoritmos para formar sonidos se programan también mediante la matriz de conexión de operadores FM.

Otra nueva capacidad de éste sintetizador es el arpegiador programable de 32 pasos. Gracias a él, podemos crear patrones de sonidos con distintas configuraciones y subdividir el teclado para que sólo cierto rango de teclas accionen el arpegiador.

[18]Las *instancias* se pueden entender como el número de veces que se emplea el sintetizador, normalmente en un proyecto de un software de producción musical. Así, por ejemplo, si se está creando una secuencia musical que contiene un bajo y un piano eléctrico interpretado desde el *FM*8, estaríamos utilizando 2 instancias de éste sintetizador.

Figura 1.19: Módulo de efectos del *Native Instruments FM*8.

El *FM*8 presenta un transformador de sonido (*sound morpher*) en la parte superior de la interfaz (véase Fig. 1.18). El *sound morpher* permite variar el sonido de acuerdo a 4 timbres seleccionados por el usuario. Cada timbre se encuentra asignado a una de las esquinas de un panel cuadrado (controlador $X - Y$), de manera que el usuario puede mediante un selector deslizante, variar la combinación de dichos timbres, dependiendo de la zona en la que sitúe dicho selector.

Para finalizar, cabría destacar el bajo coste computacional que éste sintetizador requiere para ser utilizado. En consecuencia, se aporta al usuario la gran ventaja de no necesitar un ordenador con mucha potencia para crear múltiples instancias del *FM*8. En pocas palabras, el *FM*8 constituye un software muy eficiente y versátil que mantiene la esencia de la síntesis FM.

En caso de que el lector desee escuchar una melodía de ejemplo compuesta con el sintetizador en cuestión, acceda al siguiente archivo en la carpeta adjunta `Audio`:

`5.-Native Instruments FM8-Ejemplo.mp3`

1.4.2.3. Propellerheads PX7

La empresa sueca *Propellerheads*, reconocida a nivel mundial por lanzar al mercado innovadores programas relacionados con la producción musical, dio su pincelada a finales del año 2012 cuando puso a la venta el sintetizador *PX*7, fundamentado en síntesis FM (véase Fig. 1.20). El *PX*7 se distribuye como una extensión para *Propellerheads Reason*, un famoso *software* de dicha compañía dedicado a la producción y grabación musical.

Figura 1.20: Sintetizador *PX7* para *Propellerheads Reason*.

El *PX7* consiste en un sintetizador FM de 6 operadores cuyo objetivo es recrear de manera fiel los sintetizadores clásicos de la serie *DX*, con capacidad para concebir no solo sonidos propios de los años 80, sino también actuales.

Con 32 algoritmos seleccionables, 8 parámetros de ajuste para la envolvente de cada operador, diversos macros o *faders* para variar el timbre del sonido final y escalado de teclado (entre otras características), el *PX7* se convierte en un sintetizador muy intuitivo con el que no se necesita un gran conocimiento de síntesis FM para crear complejos sonidos.

Aparte de incluir una gran cantidad de *patches* iniciales (o configuraciones preestablecidas), *Propellerheads* ha puesto al servicio del usuario, de manera gratuita, un programa conversor de *patches SysEx* del *Yamaha DX7* a *patches* para *PX7*[19].

Finalmente, puede acceder al siguiente enlace que muestra el vídeo promocional del sintetizador *PX7*:

`http://youtu.be/DiJ5cdlV34E`

[19]Sitio web del conversor:
`http://www.propellerheads.se/support_area/dx7-px7-converter/`

1.5. Historia de la síntesis FM

La síntesis FM fue desarrollada a partir de 1967 por *John M. Chowning*, compositor y doctor de la Universidad de *Stanford* (*California*). *Chowning* fue la persona que creó, apoyado en los estudios vigentes sobre radiotransmisión por frecuencia modulada[20], una eficiente herramienta que se convertiría en la síntesis digital por excelencia a finales del siglo XX.

(a) (b)

Figura 1.21: (a) *John M. Chowning*. (b) *John M. Chowning* en un estudio del CCRMA (*Center for Computer Research in Music and Acoustics*) de *Stanford*, década de los 80.

Fue en 1967 cuando *Chowning* descubrió el algoritmo de la síntesis FM, escuchando lo que sucedía cuando existía una señal moduladora que modulaba otra señal portadora, en frecuencias del rango audible (20 Hz−20 kHz). Observó cómo de una forma sencilla, alterando un número reducido de parámetros, era capaz de producir grandes modificaciones en la dinámica de las componentes espectrales (timbre) del sonido sintetizado.

Durante los seis años posteriores, *Chowning* prosiguió su investigación para instaurar las bases de éste nuevo sistema y llevarlo a la praxis. Un año más tarde, en 1973, *Chowning* publicó en el *Journal of Audio Engineering Society* la formulación de su teoría *The Synthesis of Complex Audio Spectra by Means of Frequency Modulation*, obteniendo notable repercusión tanto en medios académicos como en la industria (para más información, consulte [1]).

Chowning y sus colegas de la Universidad de *Stanford* estrecharon lazos con la compañía nipona, *Yamaha* (a diferencia de empresas americanas que no estuvieron interesadas), creando la patente en 1975 y permitiendo que dicha empresa fabricara más adelante una nueva gama de sintetizadores basados en el algoritmo de síntesis FM que *Chowning* ideó: nacía la serie *DX*.

[20] La invención de la transmisión/recepción de radio mediante frecuencia modulada (FM) se debe al ingeniero eléctrico *Edwin Howard Armstrong* (patentada en 1933), con el fin de paliar las interferencias que aparecían en la transmisión por modulación en amplitud (AM).

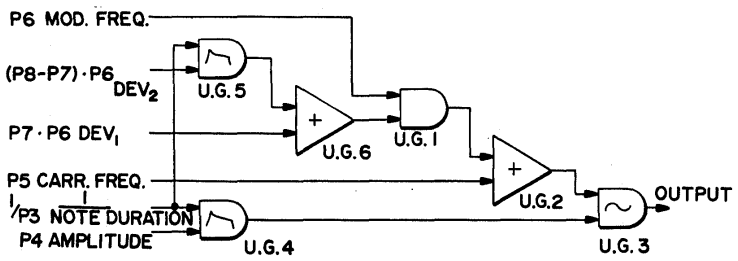


Figura 1.22: Diagrama de un instrumento FM con variación dinámica del espectro (extraído de la patente US4018121). Éste diagrama está escrito en notación MUSIC V, un programa de síntesis de sonido basado en FORTRAN y creado por *Max V. Mathews* en *Bell Telephone Laboratories* (1968). En el esquema pueden apreciarse 6 unidades generadoras (U.G): 2 generadores de envolvente (U.G.4 y U.G.5), 2 bloques sumadores (U.G.2 y U.G.6) y 2 osciladores (U.G.1 y U.G.3). La alteración dinámica del espectro se consigue gracias a que la frecuencia de la U.G.5 es variable, por ser inversamente proporcional a la duración de la nota. Por consiguiente, la amplitud de la moduladora también se irá modificando en función del tiempo y con ello, el timbre de la señal de salida.

Pero no fue hasta 1983 cuando los primeros ejemplares de la serie *DX* salieron a relucir (*DX*1, *DX*7 y *DX*9), dando el comienzo a la era digital de la síntesis[21]. Era una época en la que la mayoría de sintetizadores (por el entonces analógicos) disponibles en el mercado no se encontraban al alcance de todos debido a los disparatados precios. Con lo cual, *Yamaha* consiguió destacar introduciendo su nueva serie, muy eficiente y económica con respecto a la competencia. De hecho, el *DX*7 (del que ya se ha hablado en el apartado anterior) ha conseguido ser uno de los sintetizadores más vendidos de todos los tiempos.

Figura 1.23: Sintetizadores digitales más vendidos de la historia (de izda a dcha): *Korg M*1, *Roland D*-50 y *Yamaha DX*7. El *Korg M*1 combina síntesis sustractiva y por muestreo, el *Roland D*-50 está basado en síntesis aritmética lineal y el *Yamaha DX*7 utiliza síntesis FM.

[21] Cabe destacar que ya existían algunos sintetizadores digitales previos a la serie *DX*, como el *Fairlight CMI* (más de 25000 dólares, 1979) o el *Synclavier* (alrededor de 200.000 dólares, 1977).

Entre 1985 y 1988, *Yamaha* continuó expandiendo su serie *DX* con cuatro nuevos modelos: *DX*5, *DX*11, *DX*21 y *DX*27. Más adelante en 1991, lanzó otro otro modelo: el *SY*99, que contenía 6 operadores e incluía hasta un secuenciador (Fig. 1.24). Empleaba tanto síntesis FM avanzada como síntesis por muestreo, mejorando a los anteriores ejemplares. Finalmente, el último dispositivo de ésta compañía dedicado a FM (aunque también implementaba síntesis FS) fue el *FS*1*R*, distribuido en 1998, y en esta ocasión, en formato de módulo para ser instalado en *rack*.

Figura 1.24: Sintetizador *Yamaha SY*99.

Aunque *Yamaha* presentaba la licencia de la patente, evitando así que otras marcas pudieran desarrollar *hardware* basado en FM, la empresa *Casio* comenzó a rivalizar en 1984 con su gama *CZ*. Se trataba de sintetizadores *low cost* basados en una técnica denominada *síntesis por distorsión de fase* (*PD*, *Phase Distortion*) que producía resultados similares a la FM (ver Fig. 1.25). Otras empresas, como *Korg*, también pudieron hacerse un pequeño hueco en el mundo de la síntesis FM. *Korg* tuvo que comprar derechos de licencia a *Yamaha* a principios de la década de los 80, poder sacar al mercado los modelos *DS*-8 y 707 (ambos basados en FM).

Figura 1.25: Sintetizador *Casio CZ*-5000.

Desde 1995, tras la expiración de la patente de la Universidad de *Stanford*[22], el algoritmo de síntesis FM puede ser implementado libremente.

[22]Ésta patente ha sido la segunda más lucrativa de la Universidad de *Stanford*, otorgándole unas ganancias superiores a los 20 millones de dólares.

Han sido numerosos los grupos de música (generalmente de *pop* y *rock*) que han utilizado sintetizadores basados en FM, sobretodo en la década de los 80 y los 90. En el apartado anterior podrá observar algunas referencias.

(a) (b)

Figura 1.26: (a) *Herbie Hancock* en directo. (b) *Bobby Martin* (teclista de *Frank Zappa*) en el *live in Barcelona* (1988).

En la actualidad, la mayoría de sintetizadores presentan una circuitería sofisticada, en el sentido en que tienen suficiente potencia como para combinar varias técnicas de síntesis (aditiva, sustractiva, por muestreo, FM...etc). No obstante, desde el *FS*1*R*, no ha habido ningún fabricante que haya lanzado al mercado un sintetizador físico dedicado exclusivamente a la síntesis FM.

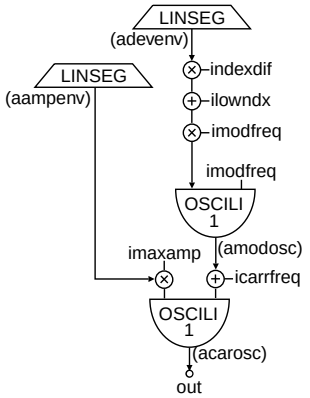
Por otra parte, desde finales de la década de los 90 hasta el día de hoy, están teniendo cada vez más repercusión los sintetizadores *software* (o virtuales). Se trata de programas que generan sonido imitando los procedimientos de síntesis de los sintetizadores *hardware*, llegando incluso a emular a modelos determinados de sintetizador. Se controlan vía MIDI, bien mediante la tarjeta de audio del ordenador, o bien a través de un interfaz USB (Fig. 1.27). Normalmente, éste tipo de *software* se distribuye como aplicación independiente o en formato *plugin*, como componente para otros programas de edición, producción o secuenciación de sonido (*VST*, *RTAS*...etc).

(a) (b) (c)

Figura 1.27: (a) Tarjeta de sonido interna (PCI). (b) Tarjeta de sonido externa (*Firewire*). (c) Interfaz MIDI-USB.

Compañías importantes como *Native Instruments* consiguieron poner a la venta en 2002 el primer sintetizador FM virtual, el *FM*7, que imitaba al *DX*7, ampliando sus capacidades y mejorando la interfaz. Cuatro años más tarde, sacaron la siguiente versión al mercado: el *FM*8. Éste sintetizador se ha convertido en una de las herramientas virtuales más potentes en lo que a tecnología de síntesis FM se refiere. A finales del 2012, la empresa sueca *Propellerheads* crea un sintetizador que, de nuevo, recupera el espíritu de la gama *DX*: el *PX*7. La única pega de éste sintetizador fue que se distribuyó como *plugin* para *Propellerheads Reason* y, que por tanto, no se podía usar de manera independiente sin dicho programa. Además de las empresas citadas, también existen otras como *Oxesoft*, *LinPlug Virtual Instruments* o *Image-Line*, que han desarrollado *software* de síntesis FM.

Finalmente, durante la década de los 90 hasta la actualidad, se han ido desarrollando lenguajes de programación orientados al cálculo de secuencias sonoras y musicales. Ejemplos de dichos lenguajes son *Csound* (Fig. 1.28), *Pure Data*, *jMax* u *Open Music*, entre otros. De este modo, el programador puede llevar a cabo el diseño de sintetizadores estableciendo configuraciones entre distintas unidades generadoras que existen en el lenguaje (osciladores, envolventes, filtros...etc). Con ello, se consigue que las restricciones a la hora de la creación del sintetizador sólo se deban a la imaginación del programador y a las especificaciones del ordenador.




```
;Definición de las variables globales reservadas.
nchnls = 1 ;Salida de audio monofónica.
sr = 44100 ;Frecuencia de muestreo.
kr = 4410 ;Frecuencia de control.
ksmps = 10 ;Muestras calculadas con los mismos valores de control.

;Instrumento FM con timbre dinámico.
instr 1
     ;Definimos las variables de tipo nota.
     inotedur = p3 ;Duración de la nota.
     imaxamp = ampdb(p4) ;Amplitud máxima de nota.
     icarrfreq = p5 ;Frecuencia de portadora.
     imodfreq = p6 ;Frecuencia de moduladora.
     ilowndx = p7 ;Límite inferior del índice de modulación.
     indexdif = p8-p7 ;Variación del índice de modulación.

     ;Definimos las variables de tipo audio (envolventes y señales).
     aampenv linseg p9, p14*p3, p10, p15*p3, p11, p16*p3, p12, p17*p3, p13
     adevenv linseg p18, p23*p3, p19, p24*p3, p20, p25*p3, p21, p26*p3, p22
     amodosc oscili (ilowndx+indexdif*adevenv)*imodfreq, imodfreq, 1
     acarosc oscili imaxamp*aampenv, icarrfreq+amodosc, 1

     ;Enviamos la señal portadora modulada a la salida de audio.
     out acarosc
endin
```

Figura 1.28: Implementación en *Csound* del instrumento FM de *Chowning* con timbre dinámico: Diagrama de bloques (izda) y código del instrumento (dcha). Ejemplo extraído de [2].

Capítulo 2

Fundamentos teóricos

En el siguiente capítulo abordaremos los contenidos teóricos necesarios para que el lector comprenda en qué consiste la síntesis de sonido, qué elementos contiene y cuáles son las principales técnicas de síntesis, tomando especial consideración en las técnicas de síntesis no lineal.

2.1. Introducción a la síntesis de sonido

Como ya se introdujo en el anterior capítulo, la *síntesis de sonido* es el proceso mediante el cual podemos generar una señal sonora, bien sea a través de medios electrónicos o calculada mediante un computador. Dicha señal resultante se denomina *señal sintetizada*. Los sistemas capaces de crear éste tipo de señales se conocen como *sintetizadores*. Son sistemas programables y están compuestos por osciladores analógicos controlados por tensión u osciladores controlados digitalmente, según se trate de un sintetizador analógico o digital, respectivamente. Además, también hemos podido comprobar que existen sintetizadores *hardware* (físicos) y sintetizadores *software* (virtuales). La señal sonora de salida podrá ser una corriente alterna (caso analógico) o un vector de números que se repetirá periódicamente en frecuencias del rango audible (caso digital). Los sintetizadores forman señales sintetizadas a partir de una o varias técnicas de síntesis. Las técnicas de síntesis podemos agruparlas en tres categorías principales: *síntesis lineal*, *síntesis no lineal* y *síntesis computacional*.

Las técnicas de síntesis lineal emplean únicamente operaciones lineales sobre las señales involucradas (suma, resta, multiplicación, filtrado lineal, etc), de modo que se pueden modificar las amplitudes, frecuencias y fases de las mismas. Por otra parte, no aparecerán nuevos parciales (componentes frecuenciales) en el espectro de la señal de salida. Pese a que son técnicas de un intuitivo control, también resultan menos eficientes en comparación con otras técnicas, como las no lineales. Ejemplos de técnicas de síntesis lineal

son la síntesis aditiva, sustractiva, cruzada o por muestreo, entre otras.

Las técnicas de síntesis no lineal utilizan operaciones no lineales sobre las señales con las que se trabaja (modulaciones o moldeado a partir de funciones de transferencia), pudiendo transformar en gran medida el contenido en parciales de dichas señales y provocando la aparición de nuevas frecuencias que antes no existían. Aunque se trata de técnicas muy eficientes, en el sentido en que generan ricos espectros con pocos recursos, también son menos intuitivas en su control, cuyos parámetros resultan más sensibles que en las técnicas de tipo lineal. Algunas técnicas de tipo no lineal son la síntesis AM, FM, PM o por moldeado de ondas.

Las técnicas de síntesis computacional están basadas en la capacidad de los ordenadores para trabajar con modelos matemáticos, ejecutando cálculos complejos que sería imposible llevar a cabo de otro modo. Es por ello que requieren de un computador programable para poder ser implementadas. La síntesis por modelos físicos o la síntesis granular son dos ejemplos ilustrativos de éste tipo de técnicas.

2.2. Elementos comunes en la síntesis de sonido

Existen diversos elementos básicos en lo que respecta a la síntesis de sonido, gracias a los cuales podemos tanto especificar el diseño de nuestro sintetizador como estudiar su comportamiento. En esta sección profundizaremos en los elementos más representativos.

2.2.1. Oscilador controlado por tensión

El oscilador controlado por tensión (VCO, *Voltage Controlled Oscilator*) constituye la columna vertebral de la síntesis de sonido. Es un dispositivo electrónico capaz de generar a su salida una señal periódica cuya frecuencia varía en función de una tensión continua de entrada (tensión de control). En el siguiente diagrama (Fig. 2.1), podemos observar un VCO que recibe tres tensiones de control: V_f, V_A y V_ϕ, que servirían para modificar el valor de la frecuencia, amplitud y fase inicial de la señal de salida $v(t)$, respectivamente.

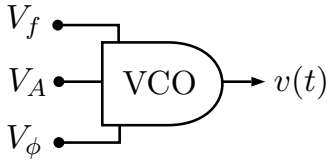


Figura 2.1: Diagrama básico de un VCO.

En el caso en que el oscilador fuese de tipo sinusoidal, obtendríamos una onda cuya expresión sería $v(t) = A\sin(2\pi ft + \phi)$, existiendo normalmente una correspondencia lineal entre las tensiones de control y los parámetros correspondientes de la señal de salida.

En los sintetizadores, cada tecla del teclado ejercía el papel de interruptor, enviando una tensión de control diferente al VCO según se pulsase una u otra. Cuando el VCO recibía dicha tensión, éste emitía una tensión de salida que oscilaba a la frecuencia correspondiente a la nota pulsada. El modo más común de realizar el control de la tensión de salida era logarítmicamente, por lo que con pequeñas variaciones de la tensión de control, provocamos grandes cambios en la señal de salida. En este sentido, si incrementamos la tensión de control en +1 V, se haría corresponder con una octava más alta en la onda resultante, +2 V para dos octavas, +4 V para tres octavas, etc. La forma de onda se creaba a partir de la repetición de un ciclo de la señal con la frecuencia seleccionada, pudiendo a menudo escoger un patrón de onda determinado (sinusoidal, cuadrada, triangular, diente de sierra, pulso...etc).

Por otra parte, tenemos el oscilador controlado digitalmente (DCO, *Digitally Controlled Oscilator*), que consiste en la implementación digital de un VCO. En esta ocasión, el DCO no es un dispositivo electrónico como tal, sino una unidad conceptual, es decir, una función de un programa cuya implementación puede realizarse de múltiples formas. Los valores de la señal de salida en un DCO pueden obtenerse bien a partir de un ciclo de la señal almacenado en una tabla de muestras, o bien calculando los valores de la salida a partir de un modelo matemático del sonido.

La problemática principal de los DCOs reside en el fenómeno del *aliasing* (solapamiento espectral). Este efecto produce distorsión en la señal generada y sucede cuando los armónicos[1] más altos de la misma superan la mitad de la frecuencia de muestreo (f_s). La distorsión de la señal de salida se produce cuando dichos armónicos se introducen en la zona de baja frecuencia del espectro. Es por ello que para evitar éste efecto indeseado se deben filtrar todos los armónicos problemáticos mediante un filtro paso bajo con frecuencia de corte f_c igual a la frecuencia de *Nyquist* ($f_c = f_s/2$). Consideremos el siguiente ejemplo para ilustrar mejor lo comentado.

Ejemplo: Deseamos sintetizar una señal diente de sierra aproximada que contenga los 5 primeros parciales y cuya frecuencia fundamental sea $f_0 = 5000$ Hz en un sistema que presenta una frecuencia de muestreo de $f_s = 44100$ Hz.

[1]Los armónicos son los múltiplos enteros de la frecuencia fundamental. Por tanto, si denotamos a la frecuencia fundamental como f_0, los armónicos de ésta frecuencia serían $2f_0$, $3f_0$, $4f_0$, etc.

El quinto parcial o armónico, de frecuencia $5f_0 = 5 \cdot 5000 = 25000$ Hz, superará la frecuencia de *Nyquist*, que en este caso será $f_s/2 = 22050$ Hz. Por consiguiente, al no "caber" dicho parcial, se reflejará en $f_s - 25000 = 19100$ Hz, distorsionado así el espectro y produciendo *aliasing*.

2.2.2. Filtro controlado por tensión

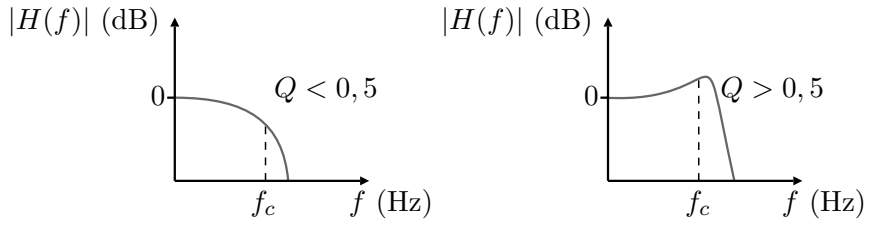
El filtro controlado por tensión (VCF, *Voltage Controlled Filter*) es otro sistema electrónico que permite realizar un filtrado a la señal que se le inyecte, por ejemplo, la procedente de un VCO. Debemos recordar que filtrar no es otra cosa sino eliminar las componentes frecuenciales que no nos interesan de una señal. El modo en el que se filtre la señal de entrada dependerá de la naturaleza o tipo del filtro: paso bajo, paso alto, paso banda, banda eliminada,...etc. Los parámetros más característicos de un VCF son la frecuencia o frecuencias de corte (f_c), el factor de calidad (Q) y su resonancia.

La frecuencia de corte del filtro determinará a partir de qué frecuencia se filtrará la señal. Si se trata de un filtro paso bajo o paso alto, bastará con especificar una frecuencia de corte (f_c); en el caso del filtro paso banda o banda eliminada, tendremos que definir 3 frecuencias de corte: la frecuencia de corte central (f_c) y dos frecuencias de corte laterales para delimitar la banda (f_1 y f_2).

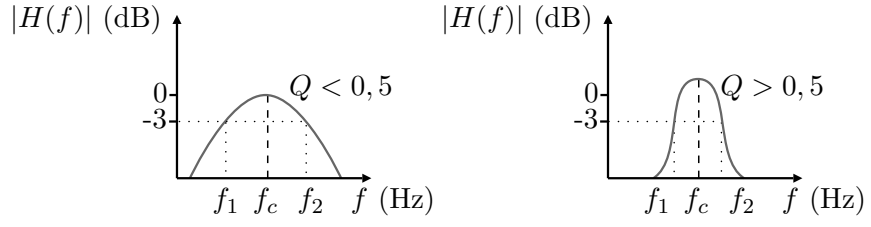
El factor de calidad (Q) determina el ancho de banda y, por tanto, cuán selectivo es el filtro. Así pues, para filtros paso banda o banda eliminada, con una Q baja, el filtro presentará un ancho de banda mayor y será menos selectivo en frecuencia (menos resonante). Por otra parte, con una Q alta, el filtro será muy selectivo y tendrá un ancho de banda pequeño (más resonante). En cambio, si el filtro es paso bajo o paso alto, cuanto mayor sea la Q, mayor será la ganancia positiva del pico centrado en la frecuencia de corte, y viceversa (Fig. 2.2).

Otro aspecto importante en lo que concierne a los VCF es la respuesta en frecuencia ($H(f)$). La respuesta en frecuencia de un filtro es una función compleja (con módulo y fase) que caracteriza el comportamiento del mismo según la frecuencia que vaya a filtrar. Si nos encontramos en el dominio del tiempo, entonces hablamos de la respuesta al impulso del filtro y se denotará como $h(t)$.

En la figura 2.3 podemos observar un esquema básico de un VCF que recibe dos tensiones de control: una para controlar la frecuencia de corte (V_{f_c}) y otra para regular el ancho de banda (V_Q). Por último, tenemos la señal de entrada $v(t)$ y la señal de salida filtrada $v'(t)$ según $H(f)$.



(a)



(b)

Figura 2.2: Efecto de Q en un filtro paso bajo (a) y paso banda (b).

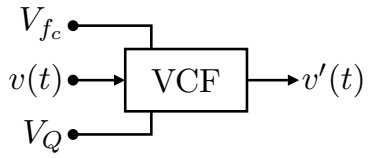


Figura 2.3: Diagrama básico de un VCF.

A diferencia de los VCF analógicos, cuya implementación puede realizarse a partir de configuraciones entre resistencias, condensadores y amplificadores operacionales (por ejemplo), también tenemos los equivalentes digitales: filtros controlados digitalmente (DCF, *Digitally Controlled Filter*). De igual modo que en los DCO, los DCF constituyen unidades funcionales que se implementan como funciones de un programa.

2.2.3. Amplificador controlado por tensión

El amplificador controlado por tensión (VCA, *Voltage Controlled Amplifier*) es un circuito electrónico que controla el nivel de la señal de salida, amplificando o atenuando la amplitud de la señal de entrada en función de una tensión de control. La figura adjunta (Fig. 2.4) muestra un esquema de un VCA. La señal de entrada $v(t)$ se amplifica dependiendo de la tensión de

control V_A, obteniendo a la salida del amplificador la señal de salida resultante: $v'(t) = A \cdot v(t)$. En ocasiones, el factor de amplificación del VCA se denomina *ganancia* (G).

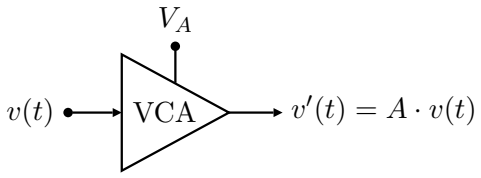


Figura 2.4: Diagrama básico de un VCA.

Idealmente, un VCA (o cualquier amplificador electrónico) no debería modificar el espectro de la señal de entrada. No obstante, un VCA real siempre introducirá ruido, inherente a todo sistema electrónico. Al incorporar ruido, se añaden nuevas frecuencias que antes no existían y, por consiguiente, alteramos el espectro de la señal de entrada.

Finalmente, resulta necesario destacar el VCA digital: el amplificador controlado digitalmente (DCA, *Digitally Controlled Amplifier*). El DCA es un operador de multiplicación que aplica un factor (> 1 para amplificar y < 1 para atenuar) sobre todos los valores de la señal de entrada. Así pues, se obtiene a la salida la misma señal pero con diferente amplitud.

2.2.4. Oscilador de baja frecuencia

El oscilador de baja frecuencia (LFO, *Low Frequency Oscilator*) es en definitiva un VCO, con la particularidad de que las señales que generan son de baja frecuencia[2], es decir, en síntesis de sonido, señales cuya frecuencia fundamental es menor que 20 Hz ($f_0 < 20$ Hz). En un LFO, normalmente podremos controlar la frecuencia, la amplitud y la forma de onda generada.

La misión principal de un LFO reside en variar de manera periódica algún parámetro de control de una señal, es decir, *modular* dicho parámetro. A la frecuencia de modulación se le denomina habitualmente *velocidad de modulación* y a la amplitud *profundidad de modulación*. La frecuencia de modulación, en estos casos, suele encontrarse en el intervalo: $0,1 < f_m < 10$ Hz.

Mediante un LFO podemos controlar cualquier módulo de un sintetizador. Analicemos brevemente algunos efectos significativos referentes a la modulación mediante LFO:

[2]En otros campos de estudio, como las telecomunicaciones, el término *baja frecuencia* se asigna a la zona del espectro electromagnético que abarca el intervalo $30 - 300$ kHz.

• *Trémolo*: Éste efecto se produce cuando modulamos periódicamente la tensión de control de la ganancia del amplificador cuando recibe una señal de entrada. En el siguiente diagrama podemos observar la implementación de éste efecto:

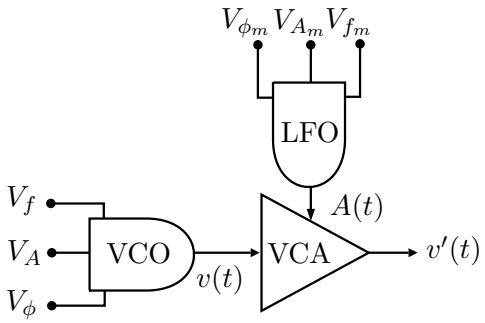


Figura 2.5: Diagrama de bloques del efecto *trémolo*.

En primer lugar, generamos una señal periódica mediante el VCO que denotamos como $v(t)$. Si consideramos que $v(t)$ es de tipo sinusoidal, la señal tendrá la forma $v(t) = A\,\mathrm{sen}(2\pi ft + \phi)$. A continuación, el VCA recibe la señal de entrada $v(t)$, que será amplificada por otra señal periódica procedente del LFO, que llamamos $A(t)$. Como vemos, se produce una modulación en amplitud de la señal de entrada $v(t)$, debido a que la ganancia del VCA ya no es un escalar, sino una función que depende del tiempo: $A(t) = A_m\,\mathrm{sen}(2\pi f_m t + \phi_m)$. Por último, obtenemos la señal de salida $v'(t)$ como el producto de ambas señales:

$$v'(t) = A(t) \cdot v(t) = [A_m \mathrm{sen}(2\pi f_m t + \phi_m)] \cdot A\,\mathrm{sen}(2\pi ft + \phi) \tag{2.1}$$

Por consiguiente, el *trémolo* no es otra cosa sino una *modulación en amplitud* cuando la frecuencia de modulación está por debajo del rango audible (< 20 Hz). En la figura 2.6 se puede observar qué está sucediendo, tanto en el dominio del tiempo como en el de la frecuencia.

En el dominio del tiempo (Fig. 2.6 izda), la señal de salida $v'(t)$ es el resultado de modular en amplitud la señal $v(t)$ mediante la función $A(t)$, produciéndose los clásicos *batidos acústicos*, donde apreciaríamos que la señal generada por el VCO ($v(t)$) aparece y se desvanece periódicamente, tantas veces por segundo como indica f_m.

En el dominio de la frecuencia (Fig. 2.6 dcha), podemos verificar que oscila la amplitud del único parcial de $v'(t)$ a la velocidad que indique f_m. Si la señal $v'(t)$ tuviera más parciales, las amplitudes de todos ellos oscilarían al unísono, nuevamente a la frecuencia que dicte f_m.

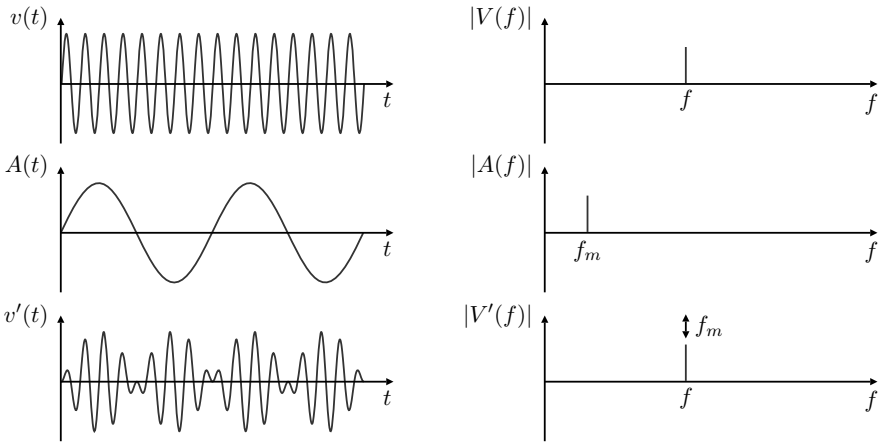


Figura 2.6: *Trémolo* en el dominio del tiempo (izda) y en el de la frecuencia (dcha).

En el siguiente archivo de audio, se ejecuta un acorde con una guitarra eléctrica y, seguidamente, se toca el mismo acorde pero con efecto *trémolo* aplicado:

6.-Trémolo.mp3

• *Vibrato*: Éste efecto se consigue modulando periódicamente la tensión de control de la frecuencia del oscilador, tal como se indica en el diagrama adjunto:

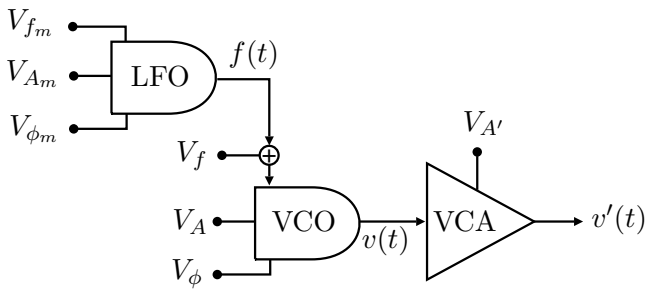


Figura 2.7: Diagrama de bloques del efecto *vibrato*.

Primeramente, el LFO genera una señal de baja frecuencia, denotada como $f(t)$. A continuación, le sumamos a dicha señal la frecuencia fundamental correspondiente a la onda sintetizada por el VCO (f). En este punto, es necesario observar que se produce una *modulación en frecuencia* en la señal creada por el VCO ($v(t)$), ya que su frecuencia fundamental (f) va a sufrir desviaciones a lo largo del tiempo en función del valor de $f(t)$.

Si consideramos el caso en el que las señales involucradas sean sinusoidales, la ecuación que caracteriza a $v(t)$ quedaría de la siguiente forma:

$$v(t) = A\,\mathrm{sen}\left[2\pi\left(f + f(t)\right)t + \phi\right] \tag{2.2}$$

$$v(t) = A\,\mathrm{sen}\left[2\pi\left(f + A_m\mathrm{sen}(2\pi f_m + \phi_m)\right)t + \phi\right] \tag{2.3}$$

Finalmente, inyectamos al VCA la señal de entrada $v(t)$, obteniendo a la salida del mismo la señal $v'(t)$. Su expresión puede verse a continuación:

$$v'(t) = A' \cdot v(t) = A' \cdot A\,\mathrm{sen}\left[2\pi\left(f + A_m\mathrm{sen}(2\pi f_m + \phi_m)\right)t + \phi\right] \tag{2.4}$$

Al producirse una modulación en frecuencia cuando f_m es menor que 20 Hz, apreciaremos cómo el sonido aumenta y disminuye su altura tan rápido como nos marque f_m. La figura 2.8 nos refleja éste hecho:

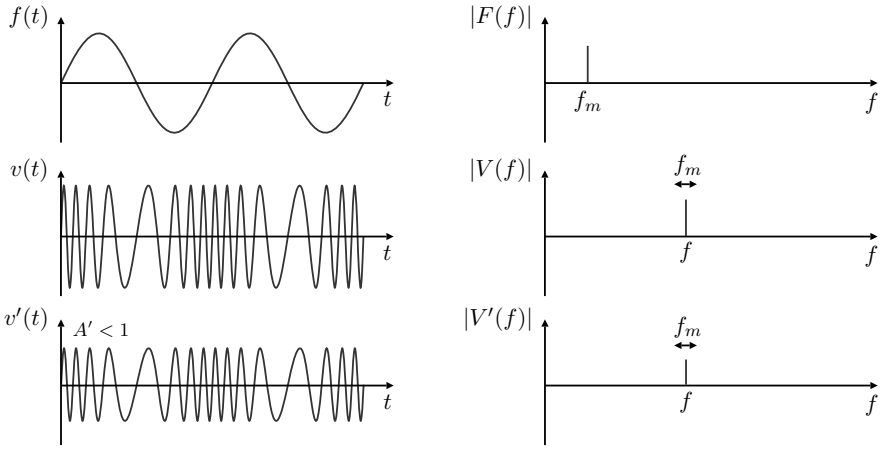


Figura 2.8: *Vibrato* en el dominio del tiempo (izda) y en el de la frecuencia (dcha).

En el dominio del tiempo (Fig. 2.8 izda), la onda generada por el VCO ($v(t)$) ya se genera modulada en frecuencia debido a la desviación frecuencial producida por el LFO ($f(t)$). Podemos fijarnos en que la frecuencia de $v(t)$ aumenta en los intervalos crecientes de $f(t)$, disminuyendo en los tramos decrecientes. En lo que respecta a la señal de salida $v'(t)$, hemos representado el caso en el que se atenúa la señal $v(t)$ por un factor de ganancia menor que la unidad ($A' < 1$).

Por otra parte, en el dominio de la frecuencia (Fig. 2.8 dcha), el único parcial de la señal de entrada $v(t)$ se desplazará a la izquierda y a la derecha periódicamente según el valor de f_m. Si $v(t)$ tuviera más parciales, todos ellos se desplazarían simultáneamente del modo mencionado.

Tomemos, a continuación, el mismo ejemplo de guitarra eléctrica que en el efecto anterior pero aplicándole un *vibrato*:

7.-Vibrato.mp3

• *Modulación de parámetros en filtros*: Debido a la gran diversidad de parámetros que podemos controlar en multitud de filtros, en esta ocasión nos vemos obligados a centrarnos en un caso particular de éste tipo de modulaciones, como es el clásico efecto *wah-wah*.

El *wah-wah* es un efecto de sonido que consiste en modificar la tensión de control de la frecuencia de corte de un filtro paso banda (f_c), transformando así el timbre de la señal de entrada. La manera en la que controlemos la frecuencia de corte puede ser manual (en el que el intérprete la modifica mediante un pedal) o bien periódica (ajustando la frecuencia de modulación mediante un potenciómetro).

El siguiente diagrama responde al caso en el que se lleva a cabo dicho efecto cuando se controla la frecuencia de corte del filtro periódicamente mediante un LFO:

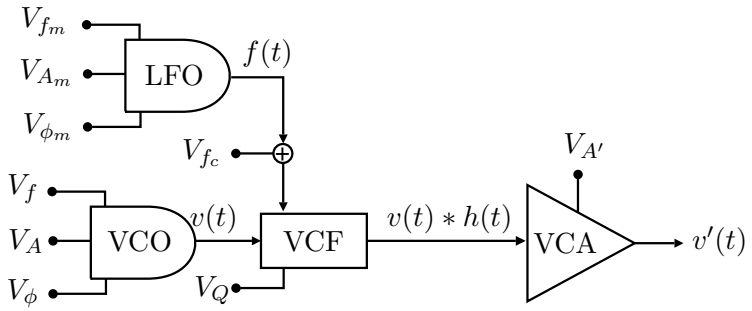


Figura 2.9: Diagrama de bloques del efecto *wah-wah* (control periódico de f_c).

El VCO se encarga de generar una señal periódica, denominada $v(t)$. Para éste caso, hemos supuesto una señal de tipo pulso con N armónicos y frecuencia fundamental f_0, cuya expresión analítica es la que sigue:

$$v(t) = A \cdot \sum_{k=1}^{N} \text{sen}(2\pi k f_0 t) \tag{2.5}$$

Seguidamente, la señal $v(t)$ se introduce en el VCF, que filtrará paso banda dicha señal, dependiendo de la respuesta al impulso $h(t)$, el factor de calidad Q y la frecuencia de corte ($f_c + f(t)$). Nótese que la frecuencia de corte del

filtro será variable, pues depende de una componente continua que representa la frecuencia de corte inicial del filtro (f_c) y de otra señal periódica producida por el LFO, que constituye la desviación en frecuencia ($f(t)$).

Por consiguiente, la modulación de la frecuencia de corte inicial de éste filtro, permitirá modificar el timbre de $v(t)$ de manera periódica, dejando pasar solo determinados armónicos o parciales en función del tiempo. En éste sentido, al dejar pasar secuencialmente los armónicos de frecuencia más baja y, a continuación, aquellos con frecuencia más alta (o viceversa), se produce un sonido característico que responde al nombre de éste efecto (*wah-wah-wah-wah...*).

Una vez se ha filtrado la señal de entrada, obtenemos a la salida del filtro la señal de entrada $v(t)$ convolucionada con la respuesta al impulso del filtro $h(t)$: $v(t) * h(t)$. En el dominio de la frecuencia, sabemos que la operación de convolución se convierte en producto, por lo que la anterior expresión quedaría como: $V(f) \cdot H(f)$. Finalmente, la señal filtrada se inyecta a un VCA que le aplicará un factor de ganancia A', obteniendo la señal que se muestra a continuación:

$$v'(t) = A' \cdot [v(t) * h(t)] = A' \cdot A \cdot \left[\sum_{k=1}^{N} \text{sen}(2\pi k f_0 t) * h(t) \right] \tag{2.6}$$

La figura (Fig. 2.10) muestra las señales involucradas en el proceso, tanto en el dominio del tiempo como en el de la frecuencia. Podemos observar que los parciales de $v(t)$ se ponderan según la respuesta en frecuencia $H(f)$, eliminando, a su vez, los que se encuentran fuera de la banda.

Además, dado que $H(f)$ se desplazará en frecuencia debido a la modulación que realiza $f(t)$, las componentes frecuenciales de la señal de salida $v'(t)$ irán cambiando periódicamente, por lo que $v'(t)$ alternará progresivamente entre las tres formas de ondas representadas para dicha señal, tan rápido como indique f_m y siguiendo el orden especificado.

Finalmente, se ha considerado el caso en el que la ganancia del VCA (A') es igual a la unidad, por lo que no se modifica la amplitud de la señal procedente del VCF.

Por último, hemos tomado la guitarra de los dos ejemplos de audio anteriores, aplicándole ésta vez el efecto *wah-wah*:

8.-Wah-wah.mp3

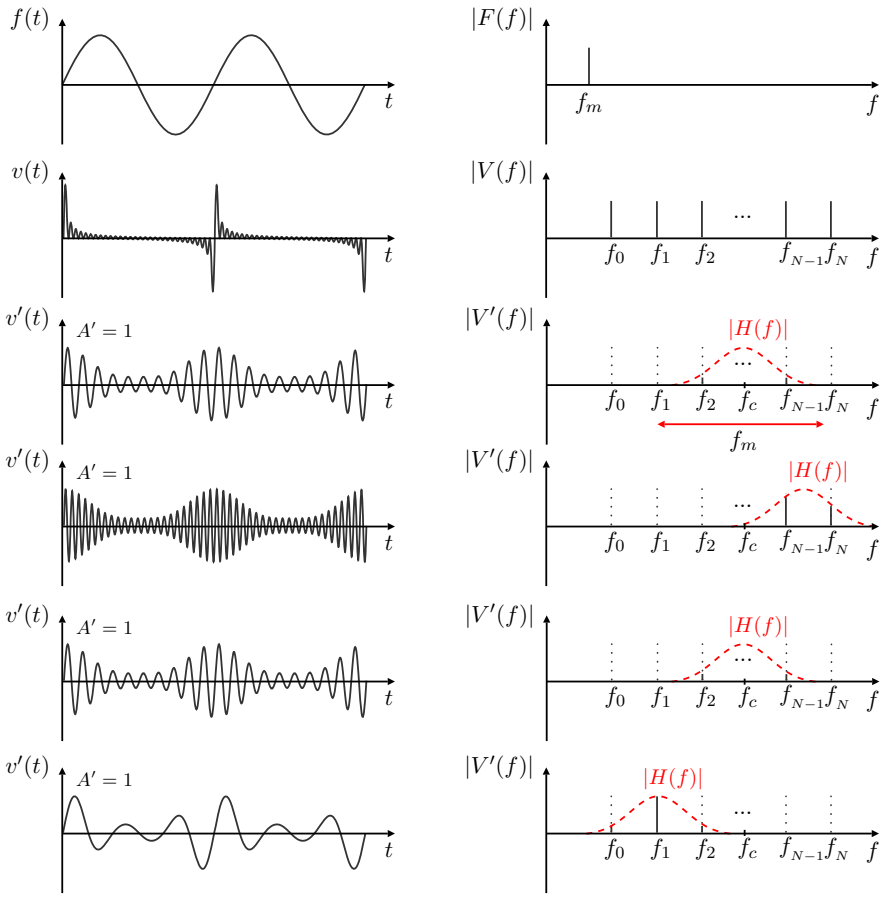


Figura 2.10: *Wah-wah* en el dominio del tiempo (izda) y en el de la frecuencia (dcha).

2.2.5. Generador de envolvente

Un generador de envolvente (EG, *Envelope Generator*) es un sistema analógico o digital que es capaz de crear una envolvente para una señal. La *envolvente* de una señal es aquella curva que nos indica la evolución de la amplitud de dicha señal en función del tiempo o de la frecuencia. Hablaremos entonces de *envolvente temporal* cuando la curva represente la amplitud frente al tiempo; y de *envolvente espectral* si la curva representa la amplitud en función de la frecuencia.

Las envolventes suelen estar formadas por la unión de diversos segmentos que nos indican las distintas fases de la misma. En función del número de segmentos que posea, podremos encontrar varios tipos de envolventes. Veamos los más significativos.

• *Envolventes ASR*:

Éste tipo de envolventes son las más sencillas y deben su nombre a las tres fases que presentan: *Attack* (A), *Sustain* (S) y *Release* (R) (Ataque, Sostenimiento y Relajación, respectivamente). La fase de ataque se controla mediante el *tiempo de ataque* (t_A) y se define como el tiempo transcurrido entre que se inicia la nota y se alcanza una amplitud estable. La fase de sostenimiento es el intervalo de tiempo en el cual la amplitud es estable y viene definida mediante la *amplitud de sostenimiento* (A_S). Finalmente, la fase de relajación se gobierna mediante el *tiempo de relajación* (t_R) y consiste en el tiempo transcurrido desde el final de la fase de sostenimiento hasta que la amplitud llega a cero. Los dos primeros parámetros se aplican cuando la nota permanece activada (*nota on*), mientras que el último parámetro tiene lugar cuando se desactiva la nota (*nota off*). En la siguiente figura, podemos observar tanto el diagrama básico de un generador de envolvente ASR, como la forma general de ésta envolvente.

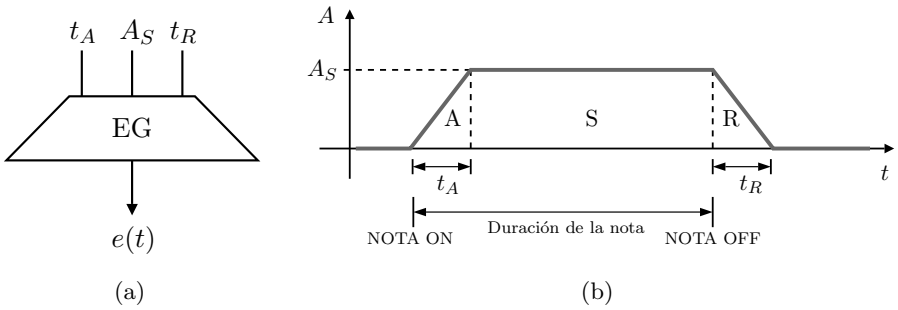


Figura 2.11: (a) Diagrama básico de un generador de envolvente ASR. (b) Forma general de una envolvente ASR.

Si no existiese fase estable (S), obtendríamos una envolvente triangular denominada AR, típica en muchos sonidos percusivos. Por consiguiente, dicha envolvente únicamente presentaría las fases de ataque y relajación, tal como muestra la Fig. 2.12.

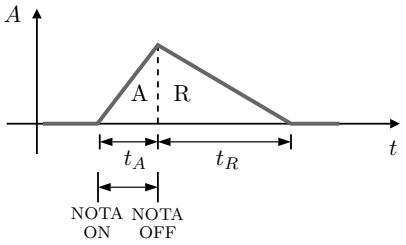


Figura 2.12: Forma general de una envolvente AR.

• *Envolventes ADSR*:

Son las envolventes más comunes que emplean los sintetizadores. La envolvente ADSR es una ampliación de la ASR, pues incluye una nueva fase de decaimiento denominada *Decay* (D). Éste nuevo tramo se puede definir como el tiempo transcurrido desde que el sonido alcanza su máxima amplitud hasta que se estabiliza. La fase de decaimiento se controla mediante el tiempo de decaimiento (t_D). Éste tipo de envolventes aparecen de manera natural en las señales procedentes de muchos instrumentos de viento, como por ejemplo cuando se ejecuta una nota en la flauta (véase Fig. 2.13).

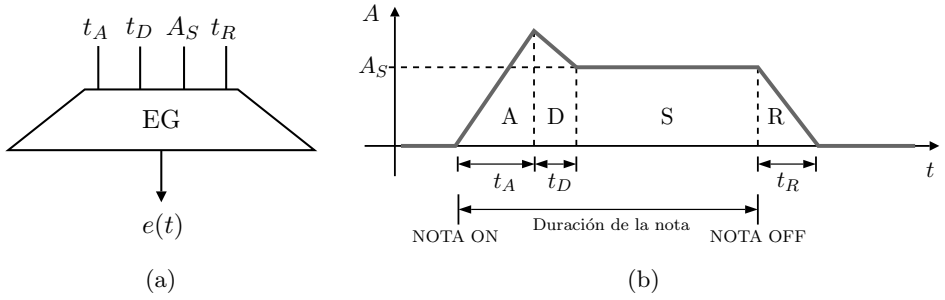


Figura 2.13: (a) Diagrama básico de un generador de envolvente ADSR. (b) Forma general de una envolvente ADSR.

Aunque los tres ejemplos anteriores constituyen algunas de las envolventes más comunes, éstas no dejan de ser casos concretos de curvas, que en general, pueden tomar cualquier forma. Los segmentos o tramos que conforman las envolventes suelen ser de dos tipos: rectilíneos o exponenciales. Cuando los tramos son rectilíneos, percibiremos una sonoridad de tipo logarítmica, mientras que si son exponenciales, notaremos una sonoridad de tipo lineal (Fig. 2.14).

Un aspecto decisivo a la hora de diseñar una envolvente, reside en evitar cambios bruscos en la mismas, y por tanto, no emplear segmentos verticales, sobretodo al principio y al final de las notas. Éstos cambios abruptos generan la aparición de *clicks* en los extremos de la envolvente, distorsionando el sonido final. Por consiguiente, no resultará apropiado establecer a cero los tiempos correspondientes a las fases de la envolvente ($t_A = t_D = t_R = 0$). En el ejemplo de la Fig. 2.15 se muestra una envolvente mal diseñada y posteriormente corregida, a fin de eludir los *clicks* al inicio y al final del sonido.

Del mismo modo que sucede con los LFO, las envolventes creadas por los EG pueden actuar sobre los distintos parámetros de los módulos de un sintetizador (osciladores, amplificadores y/o filtros).

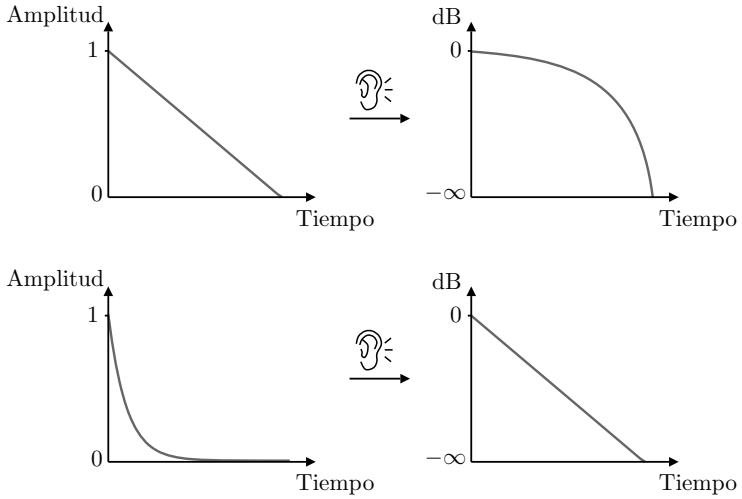


Figura 2.14: Segmentos comunes de una envolvente. A la izquierda se muestra la amplitud normalizada; a la derecha la correspondiente amplitud percibida en dB.

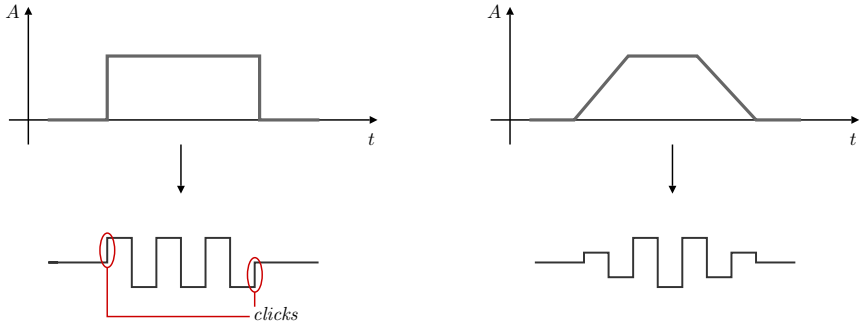


Figura 2.15: Diseño incorrecto de una envolvente (izda) y corrección de la misma (dcha), así como los efectos producidos en una onda cuadrada.

2.2.6. Tipos de señales

Sabemos que una *señal* no es otra cosa sino una magnitud física que varía a lo largo del tiempo. En síntesis de sonido, se establece una clasificación de dichas señales basada en los siguientes criterios:

1. En función del conjunto de valores que pueden tomar.
2. En función de la frecuencia fundamental.

Existen dos tipos de señales según el conjunto de valores que puedan tomar: *unipolares* y *bipolares*.

Las señales *unipolares* son aquellas que únicamente pueden variar en una mitad del rango de amplitudes. Por consiguiente, éste tipo de señales siempre serán solo positivas o solo negativas (incluyendo el valor 0). Un ejemplo de éste tipo de señales podría ser una envolvente como las ya vistas, o en general, cualquier señal que controle de manera dinámica algún parámetro.

Las señales *bipolares* serían aquellas que pueden variar en el rango completo de amplitudes. En éste sentido, las señales bipolares pueden tomar valores positivos, negativos o nulos. Ejemplos de señales bipolares podrían ser ondas periódicas (señal sinusoidal, triangular,...etc).

Una característica común tanto a las señales unipolares como bipolares sería que ambas se suelen definir *normalizadas*, esto es, que toda la señal está dividida por su valor máximo, de manera que si es una señal unipolar, ésta variará entre 0 y 1, mientras que si es bipolar tomará valores entre -1 y 1. El motivo de normalizar las señales es precisamente para poder escalarlas después al rango deseado en cada momento, multiplicando toda la señal por un escalar. En la siguiente figura podemos ver un ejemplo de señal tanto unipolar como bipolar:

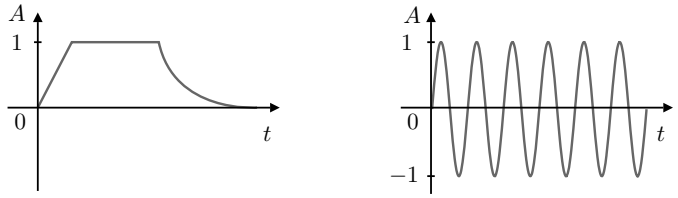


Figura 2.16: Ejemplo de señal unipolar (izda) y bipolar (dcha).

Las señales también podemos agruparlas según su frecuencia fundamental. A tales efectos, podemos distinguir dos tipos de señales: *señales de audio* y *señales de baja frecuencia*. Las señales de baja frecuencia se caracterizan por presentar una frecuencia fundamental que se encuentra por debajo del rango audible ($f_0 < 20$ Hz), mientras que el resto se considerarían señales de audio. Las señales de baja frecuencia son útiles para que actúen como moduladoras de otras señales, variando periódicamente parámetros audibles de dichas señales. La importancia de éste tipo de señales ya se vio en el apartado correspondiente al LFO.

2.2.7. Formas de onda

A lo largo de éste capítulo, hemos podido comprobar que algunos dispositivos tales como los VCO o los LFO, son capaces de generar distintas señales periódicas. Dichas señales pueden presentar una forma de onda característica, que frecuentemente, podrá ser seleccionada desde el propio oscilador que

la genera. A continuación, realizaremos un breve análisis de las formas de onda más características en síntesis de sonido.

- SEÑAL SINUSOIDAL: Constituye una de las señales principales en síntesis de sonido. A partir de ella, podemos construir otras formas de onda más complejas, aplicando el desarrollo en serie de *Fourier*[3]. La ecuación en el dominio del tiempo, así como la forma de onda y su espectro, ya ha podido observarse en secciones anteriores del presente capítulo.

- SEÑAL CUADRADA: Ésta señal contiene únicamente los armónicos o parciales impares, con amplitudes que decaen proporcionalmente a $1/n$, siendo n el número del parcial. Por tanto, la ecuación de síntesis sería la que sigue:

$$x(t) = A \cdot \sum_{n=1}^{N} \left[\frac{1}{2n-1} \cdot \operatorname{sen}\left(2\pi(2n-1)f_0 t\right) \right] \tag{2.7}$$

  Cuanto mayor sea el número total de armónicos N que empleemos en la construcción de la onda, obtendremos una mejor aproximación, es decir, una señal que se parezca más a una onda cuadrada ideal. En la siguiente figura, podemos apreciar la forma de onda de una señal cuadrada, así como su espectro:

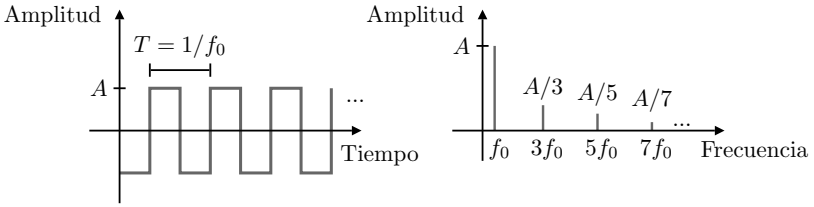


Figura 2.17: Forma de onda de una señal cuadrada (izda) y su espectro correspondiente (dcha).

- SEÑAL TRIANGULAR: Se forma mediante la suma de los parciales impares, pero ésta vez decaen de manera proporcional a $1/n^2$. La expresión para sintetizar ésta señal se puede ver a continuación:

[3]El *Teorema de Fourier* indica que cualquier señal periódica puede descomponerse en una combinación lineal infinita de funciones sinusoidales simples:

$$x(t) = \sum_{n=-\infty}^{+\infty} A_n \cdot \operatorname{sen}\left(2\pi f_n t + \phi_n\right)$$

$$x(t) = A \cdot \sum_{n=1}^{N} \left[\frac{1}{(2n-1)^2} \cdot \text{sen}\,(2\pi(2n-1)f_0 t) \right] \tag{2.8}$$

Cuando las fases de los armónicos son iguales, se obtiene una onda que se aproxima más bien a una de tipo sinusoidal. Para evitar ésto y conseguir una forma triangular, se invierte la fase de los armónicos múltiplos de 3. Recordemos que invertir la fase de un parcial equivale a expresar dicho parcial con su amplitud cambiada de signo:

$$A{\cdot}\text{sen}(2\pi f_0 t{+}\phi{+}\pi) = A{\cdot}\text{sen}(-(2\pi f_0 t{+}\phi)) = -A{\cdot}\text{sen}(2\pi f_0 t{+}\phi) \tag{2.9}$$

Seguidamente, se presenta una forma de onda triangular (con los parciales correspondientes invertidos de fase) y el espectro adjunto:

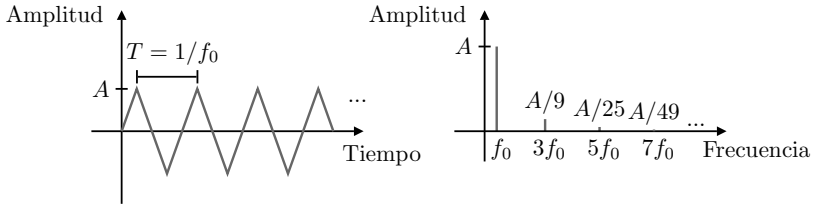


Figura 2.18: Forma de onda de una señal triangular (izda) y su espectro correspondiente (dcha).

- SEÑAL DIENTE DE SIERRA o RAMPA: Presenta todos los armónicos con amplitudes que decaen proporcionalmente a $1/n$. La ecuación para construir una onda tipo diente de sierra se muestra a continuación:

$$x(t) = A \cdot \sum_{n=1}^{N} \left[\frac{1}{n} \cdot \text{sen}\,(2\pi n f_0 t) \right] \tag{2.10}$$

Tanto la forma de onda como el espectro correspondiente se muestran en la siguiente figura:

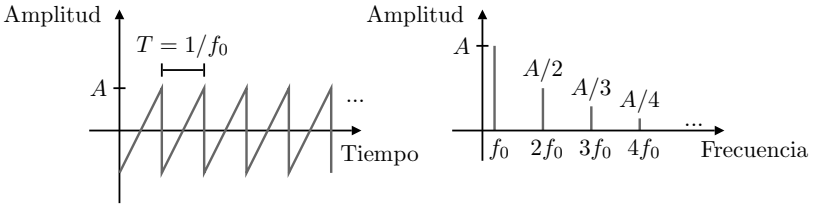


Figura 2.19: Forma de onda de una señal diente de sierra (izda) y su espectro correspondiente (dcha).

- SEÑAL PULSO: La característica de ésta señal reside en que contiene todos los armónicos con la misma amplitud. Puede ser sintetizada a partir de la siguiente expresión:

$$x(t) = A \cdot \sum_{n=1}^{N} \text{sen}\,(2\pi n f_0 t) \tag{2.11}$$

Veamos el aspecto de dicha señal tanto en el dominio del tiempo como en el de la frecuencia:

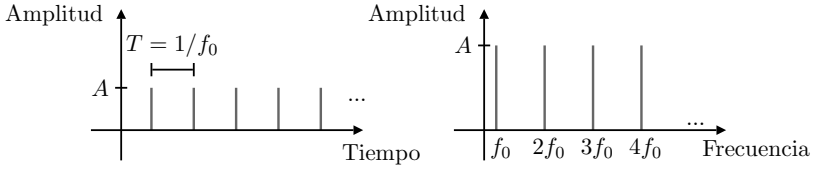


Figura 2.20: Forma de onda de una señal pulso (izda) y su espectro correspondiente (dcha).

- SEÑAL RECTANGULAR: Presenta únicamente aquellos armónicos que no son múltiplos de la inversa del ciclo activo $(1/c)$. Las amplitudes de dichos armónicos decaen de manera proporcional a $1/n$.

El *ciclo activo* (o *ciclo de trabajo*) de la señal rectangular es la relación entre el tiempo que dura el nivel de máxima amplitud en un ciclo (a) y la duración del periodo de la señal (T). Analíticamente:

$$c = \frac{a}{T} \tag{2.12}$$

La ecuación que nos permite obtener la señal rectangular en el dominio del tiempo es la siguiente:

$$x(t) = A \cdot \sum_{n=1}^{N} \left[\frac{1}{n} \cdot \text{sen}\,(2\pi n f_0 t) \right] \qquad (n \neq 1/c) \tag{2.13}$$

La figura 2.21 ilustra un ejemplo de señal rectangular con ciclo activo $c = 1/4$, tanto en el dominio del tiempo como en el de la frecuencia. Como podemos observar, los armónicos múltiplos de 4 no aparecerían.

Para finalizar el apartado, acceda al siguiente archivo si desea escuchar una secuencia de éstas señales ordenadas según se han visto:

9.-Formas de onda.mp3

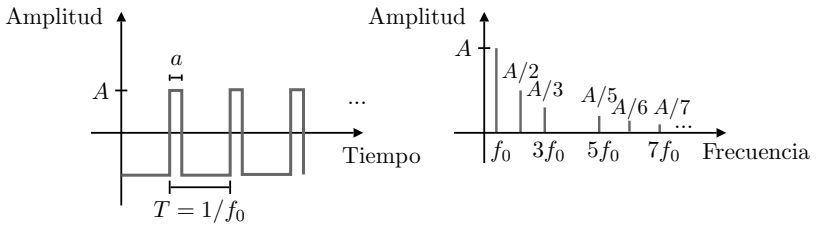


Figura 2.21: Forma de onda de una señal rectangular (izda) y su espectro correspondiente (dcha) para $c = 1/4$.

2.2.8. Configuración de elementos

La aparición de los primeros lenguajes de programación de síntesis (década de los 90) conllevó la necesidad de especificar los algoritmos de síntesis que eran diseñados. En éste sentido, se creó una notación gráfica que debía ilustrar el funcionamiento de los sintetizadores creados, de manera que se pudiera implementar en cualquier sistema, independientemente del lenguaje empleado.

En apartados anteriores, hemos visto las representaciones gráficas de los elementos básicos que intervienen en la síntesis de sonido. Pudimos observar que los osciladores se representan mediante el símbolo de la puerta lógica AND, los amplificadores mediante triángulos, los filtros mediante rectángulos y los generadores de envolventes mediante trapecios (véanse Fig. 2.1, 2.3, 2.4 y 2.13).

Todos éstos elementos (así como muchos otros), pueden conectarse entre sí mediante líneas que representan el flujo de información, formando una *configuración de elementos*. El flujo de información se transmitiría de arriba a abajo y de izquierda a derecha. Los valores *constantes* se representarían con líneas, mientras que las *señales* (valores variables) las representamos mediante flechas.

Por ejemplo, en la Fig. 2.3, podemos observar que los valores de frecuencia de corte y de resonancia (V_{f_c} y V_Q) que se introducen a la entrada del filtro son constantes, por lo que dibujamos la conexión mediante una línea. En cambio, tanto la señal que se inyecta a la entrada como la de salida ($v(t)$ y $v'(t)$), al ser variables, las representaríamos con flechas.

Además, a todas las conexiones o controles se les pueden aplicar operadores matemáticos (sumadores, restadores, multiplicadores, inversores...etc) para satisfacer nuestras especificaciones de diseño. En la siguiente figura se muestran algunos operadores básicos:

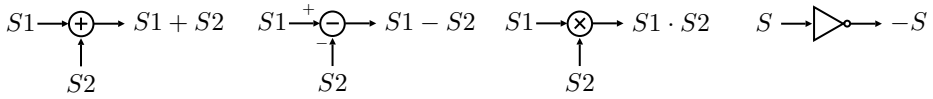


Figura 2.22: Operadores matemáticos aplicables a valores constantes y/o señales. De izquierda a derecha: sumador de dos señales, restador de dos señales, multiplicador de dos señales e inversor de una señal.

Aunque en la anterior figura se han ilustrado únicamente operadores que aceptan una o dos señales (operadores unarios y binarios), pueden existir operadores que acepten un mayor número de señales. Por otra parte, éstos operadores también serían válidos para valores constantes, así como una combinación de valores constantes y señales. Por ejemplo, un amplificador podría verse como un operador que multiplica una señal por un valor constante (ganancia).

A continuación, veremos algunos ejemplos de configuraciones de síntesis a partir de conexiones de elementos básicos, analizando el funcionamiento del mismo modo que se hizo en la descripción de los efectos mediante LFO (véase sección 2.2.4).

Ejemplo 1: Oscilador sinusoidal controlado por envolvente

El siguiente diagrama mostraría un oscilador sinusoidal cuya amplitud está gobernada por un generador de envolvente de tipo ADSR:

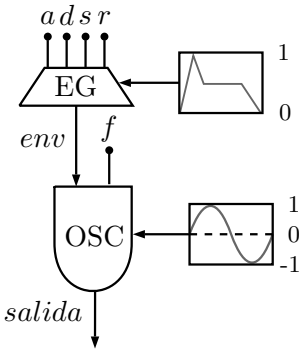


Figura 2.23: Diagrama de bloques de un oscilador sinusoidal con envolvente ADSR.

En primer lugar, el generador de envolvente EG construye una envolvente de amplitud de tipo ADSR normalizada que denominamos *env*, a partir de los parámetros a, d, s y r, así como la curva formada por los distintos segmentos. La señal envolvente controlará ahora la amplitud del oscilador OSC a lo largo del tiempo. A continuación, el oscilador OSC genera una señal sinu-

soidal de salida (*salida*), repitiendo periódicamente (a una frecuencia f) la tabla normalizada introducida que contiene un ciclo de una onda sinusoidal. Finalmente, podemos observar qué aspecto tendrá la señal *salida*:

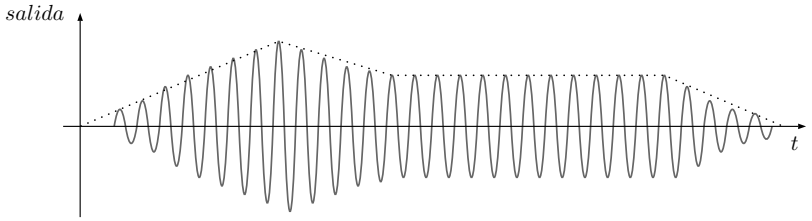


Figura 2.24: Representación de la señal de salida del oscilador (*salida*).

Ejemplo 2: Instrumento FM básico

En éste ejemplo se presenta una posible implementación de un instrumento sencillo basado en síntesis FM haciendo uso de dos osciladores:

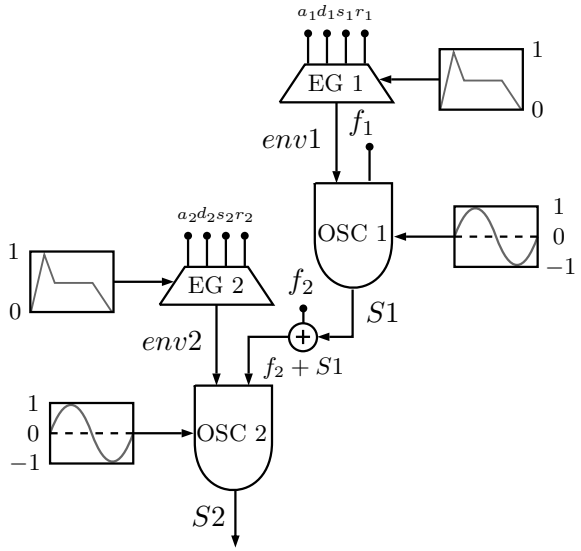


Figura 2.25: Diagrama de bloques de un instrumento FM básico.

La función principal de ésta configuración consiste en modificar la frecuencia de la señal de salida de un oscilador (OSC 2) a través de otro (OSC 1) cuando la frecuencia de éste último se encuentra en el rango audible ($f_1 > 20$ Hz), provocando la aparición de nuevos parciales en la señal de salida $S2$. De éste modo se lleva a cabo un tipo de síntesis denominada *síntesis FM*, sobre la que posteriormente indagaremos en profundidad.

Comenzamos creando una envolvente mediante el generador EG 1 de tipo ADSR ($env1$), que modificará dinámicamente la amplitud del primer oscilador (OSC 1) a lo largo del tiempo. Del mismo modo que en el ejemplo anterior, introducimos los parámetros correspondientes a los tiempos de las distintas fases de la envolvente (a_1, d_1 y r_1), la amplitud de sostenimiento (s_1) y una tabla normalizada con la curva correspondiente. Por otra parte, se introduce un valor constante de frecuencia (f_1) y el ciclo normalizado de onda sinusoidal, generando la señal de salida del primer oscilador: $S1$. Cabe destacar que si f_1 está por debajo de 20 Hz, se produciría un vibrato, aunque no es el cometido de ésta especificación.

A continuación, mediante un bloque sumador, se suma la frecuencia fundamental del segundo oscilador (f_2) con la señal de salida del primer oscilador ($S1$), de modo que ahora la frecuencia de OSC 2 será variable en el tiempo, y además, la señal $env1$ ofrecerá mayor dinámica al timbre final. En éste momento, generamos la segunda envolvente ($env2$) mediante el bloque EG 2, siguiendo el mismo procedimiento que con el anterior oscilador. La diferencia entre ésta envolvente y la anterior, reside en que $env2$ actuará en el dominio del tiempo sobre la señal de salida, mientras que $env1$ lo hará en frecuencia. Añadiendo finalmente la tabla con el ciclo de onda sinusoidal, obtendríamos la señal de salida del segundo oscilador $S2$.

Ejemplo 3: Sintetizador aditivo con control de reverberación

En la figura 2.26 podemos observar un ejemplo propuesto de sintetizador aditivo que presenta un control para añadir reverberación. Éste sintetizador se fundamenta en *síntesis aditiva*. La síntesis aditiva se basa en la creación de ondas complejas a partir de la superposición de ondas simples, siguiendo el *Teorema de Fourier*.

Primeramente, se construye una envolvente (env) de tipo ADSR mediante el generador EG, que controlará la amplitud a lo largo del tiempo de los cuatro osciladores simultáneamente (OSC 1, OSC 2, OSC 3 y OSC 4). A continuación, introducimos las frecuencias de cada oscilador (f_1, f_2, f_3 y f_4) y las tablas con los ciclos de las ondas correspondientes: sinusoidal, cuadrada, triangular y diente de sierra.

Generadas las cuatro señales ($S1$, $S2$, $S3$ y $S4$), se amplifica cada una mediante el amplificador correspondiente, lo que permitirá modificar el nivel de cada una en la futura mezcla. A continuación, obtenemos la señal de salida S a partir de la suma de las cuatro señales amplificadas. Esto es:

$$S = S1 \cdot G1 + S2 \cdot G2 + S3 \cdot G3 + S4 \cdot G4 \tag{2.14}$$

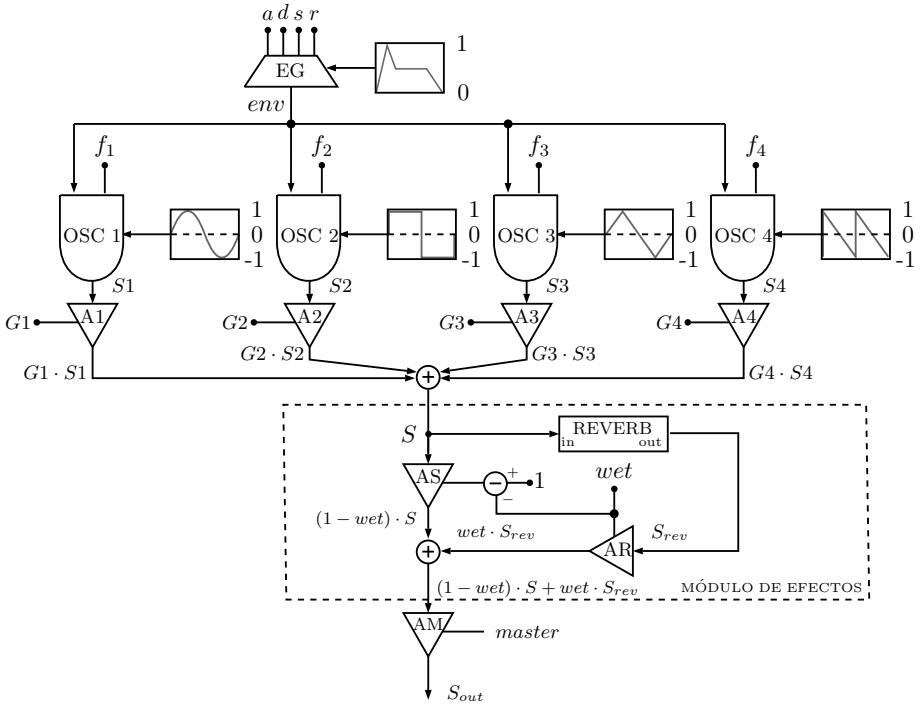


Figura 2.26: Diagrama de bloques de un sintetizador aditivo con control de reverberación.

Llegados a éste punto, nos encontramos con el módulo de efectos, que permitirá al usuario añadir una cantidad de reverberación a la señal de salida. Como vemos, la señal S se inyecta, por una parte, al amplificador AS y por la otra, se envía a la entrada del dispositivo de reverberación, obteniendo a la salida del mismo la señal S con reverberación añadida, es decir, S_{rev}.

Seguidamente, la señal con reverberación se introduce en el amplificador AR, amplificando dicha señal por el valor de ganancia *wet*, comprendido entre 0 y 1. Por consiguiente, a la salida de éste amplificador obtendremos la cantidad de señal con reverberación que deseamos ($wet \cdot S_{rev}$), que se introducirá en un bloque sumador. Nótese que el parámetro *wet* no solo amplifica a la señal reverberada, sino que también se envía a un bloque restador, para restarse a la unidad $(1 - wet)$. El factor $(1 - wet)$ amplificará a la señal sin efecto (seca o *dry*) que denotamos como S, que también se inyectará al bloque sumador con la señal reverberada.

A la salida del bloque sumador, conseguiremos una mezcla entre señal seca y señal reverberada:

$$(1 - wet) \cdot S + wet \cdot S_{rev} \qquad 0 \leq wet \leq 1 \tag{2.15}$$

Como podemos ver, el parámetro *wet* presenta doble función, puesto que controla la cantidad tanto de señal reverberada como de señal seca. Si nos fijamos en la anterior expresión, cuando no deseamos reverberación, estableciendo *wet* a cero ($wet = 0$), quedaría únicamente en la mezcla final la señal sin efecto (S). En cambio, si deseáramos una señal completamente reverberada, estableciendo *wet* a la unidad ($wet = 1$), en la mezcla final solo existiría señal con reverberación (S_{rev}).

Finalmente, la mezcla final se manda a un último amplificador (AM) que controlará el nivel general de la señal de salida del sintetizador a través del parámetro de ganancia ajustable *master*:

$$S_{out} = master \cdot [(1 - wet) \cdot S + wet \cdot S_{rev}] \tag{2.16}$$

En éstos tres ejemplos hemos podido comprobar cómo una notación gráfica no solo facilita el entendimiento de los algoritmos de síntesis, sino que nos permite construir especificaciones tan complejas como queramos. Las configuraciones diseñadas constituyen el paso previo a la implementación específica en un lenguaje de programación concreto. Por tanto, será necesario adaptar cada elemento del diagrama a la sintaxis del lenguaje empleado.

2.3. Síntesis no lineal

Al principio del capítulo pudimos ver que las técnicas de *síntesis no lineal* estaban basadas en operaciones que podían alterar el contenido espectral de las señales originales, permitiendo que apareciesen nuevos parciales donde antes no había, a diferencia de los métodos de síntesis lineales.

Las técnicas de síntesis no lineal presentan diversas *ventajas* que es necesario tenerlas presente:

- Son técnicas muy eficientes, puesto que podemos generar sonidos con timbres complejos a partir de pocos osciladores, controlando, por tanto, pocos parámetros.
- Presentan un bajo coste computacional, por lo que se simplifica la circuitería en la implementación del sintetizador.
- Se consigue un espacio tímbrico característico.

Por otra parte, las técnicas de síntesis no lineal también presentan ciertos *inconvenientes*:

- El control del espectro resulta poco intuitivo debido a que los parámetros con los que se modifica son muy sensibles. Esto significa que pequeñas variaciones en los parámetros de control provocan grandes cambios en el timbre.

- Las técnicas de síntesis no lineales no resultan muy aptas para imitar sonidos naturales (síntesis imitativa).

Éste tipo de síntesis empezó a cobrar especial importancia a partir de la década de los 80, momento en que como ya vimos, aparecieron los primeros sintetizadores digitales no lineales en el mercado, ofreciendo una alternativa a los sintetizadores analógicos sustractivos del entonces[4].

En lo que respecta a los tipos de síntesis no lineal, podemos destacar dos grandes grupos: la síntesis por modulación y la síntesis por moldeado de ondas (*waveshaping*).

La *síntesis por modulación* está basada en la modificación de una señal (*portadora*) mediante la característica de otra señal (*moduladora*). En éste sentido, existen múltiples técnicas de síntesis por modulación: síntesis por modulación en amplitud (AM), en frecuencia (FM), en fase (PM), en anillo (RM)...etc. Ya hemos podido estudiar algunos casos de modulación sonora, como el *trémolo* y *vibrato* (modulación en baja frecuencia de amplitud y frecuencia, respectivamente) o incluso una modulación FM en el Ejemplo 2 del anterior apartado.

En la *síntesis por moldeado* únicamente se utiliza un oscilador que genera una señal periódica (*excitadora*), que será *moldeada* por una función matemática (*función de moldeado*). Ésta funcion (generalmente no lineal) recibe a su entrada la señal periódica (normalmente sinusoidal), convirtiéndola a su salida en la señal deseada. Al emplear un único oscilador, se pone de manifiesto la eficiencia de la síntesis por moldeado, permitiendo generar formas de onda muy complejas. Ejemplos de técnicas de síntesis por moldeado serían el *waveshaping* o el *wave terrain*.

Debido a la gran cantidad de bibliografía referente a las técnicas de síntesis tanto lineal como no lineal, profundizaremos únicamente en la *síntesis por modulación en frecuencia* (FM), puesto que el sintetizador del que trata el presente proyecto está basado en ella. Si desea indagar en otras técnicas de síntesis, consulte [5] y [6].

2.3.1. Síntesis por modulación en frecuencia (FM)

La *síntesis por modulación en frecuencia* (FM) empezó a ser conocida gracias a los primeros sintetizadores comerciales con osciladores digitales de la empresa *Yamaha* (serie *DX*), que como pudimos ver en el capítulo anterior, estaban basados en dicha técnica, desarrollada por *J.M. Chowning*.

[4]Debido a la inestabilidad de los sistemas analógicos, no fue adecuada la implementación de las técnicas no lineales, por lo que fueron los sintetizadores digitales los que comenzaron a permitir implementaciones fiables.

Cuando variamos la frecuencia de una señal portadora a través de otra señal moduladora cuya frecuencia es superior a 20 Hz ($f_m > 20$ Hz o $T_m < 50$ ms), se produce una modificación en el timbre de la portadora al añadirse nuevos parciales en el espectro de dicha señal. Por consiguiente, la gran ventaja de la síntesis FM reside en que con muy pocos osciladores podemos conseguir señales con espectros muy ricos en parciales. En pocas palabras, es una síntesis eficiente.

Si consideramos como señal portadora (*carrier signal*) la siguiente onda sinusoidal:

$$s_c(t) = A_c \cdot \text{sen}\,(2\pi f_c t + \phi_c) \tag{2.17}$$

Y como señal moduladora (*modulating signal*) la que se muestra a continuación:

$$s_m(t) = A_m \cdot \text{sen}\,(2\pi f_m t + \phi_m) \tag{2.18}$$

La ecuación que gobierna la modulación FM para ambas señales, tomando fase iniciales nulas ($\phi_c = \phi_m = 0$ rad), quedaría de la siguiente forma:

$$s(t) = A_c \cdot \text{sen}\,[2\pi f_c t + A_m \text{sen}(2\pi f_m t)] \tag{2.19}$$

Por otro lado, la configuración correspondiente a dicha modulación se ilustra en la figura adjunta:

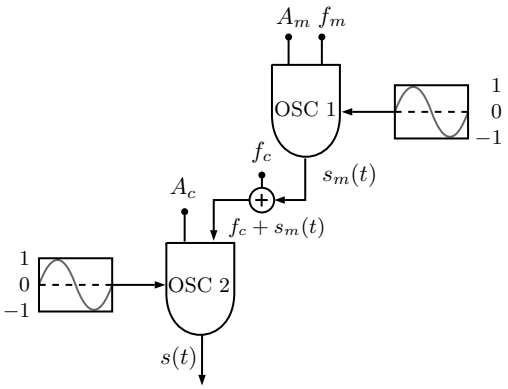


Figura 2.27: Diagrama básico de la síntesis por modulación en frecuencia.

Fijándonos en la expresión 2.19, observemos que ahora la frecuencia de la señal $s(t)$ será variable a lo largo del tiempo y tendrá un valor de $f_c + A_m \text{sen}(2\pi f_m t)$.

Cuando la señal moduladora toma valores nulos ($s_m(t) = 0$), la frecuencia de la señal $s(t)$ será f_c. Por otro lado, si la moduladora alcanza su valor máximo (A_m) o su valor mínimo ($-A_m$), la frecuencia de la señal $s(t)$ valdrá $f_c + A_m$ o $f_c - A_m$, respectivamente. Todo ésto se vé reflejado en la Fig. 2.28.

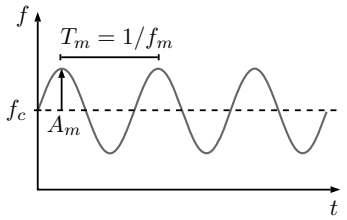


Figura 2.28: Representación de la frecuencia de la señal de salida $s(t)$ en función del tiempo.

Al valor de amplitud de la moduladora (A_m) también se le suele denominar *desviación*, que se mide en Hz por sumarse a otro valor de frecuencia.

El *índice de modulación* es un parámetro muy importante en lo referente a la síntesis FM y se define como la relación entre la amplitud y la frecuencia, ambas de la moduladora, es decir:

$$I = \frac{A_m}{f_m} \tag{2.20}$$

Éste parámetro es adimensional (sin unidades), ya que en éste caso, tanto amplitud como frecuencia están en Hz. Más adelante veremos que el índice de modulación resulta fundamental para controlar la riqueza espectral del sonido generado.

Por último, cabe destacar que cuando el índice de modulación varía a lo largo del tiempo, hablaremos de *envolvente del indice de modulación* y la representaremos como $I(t)$. En la práctica, para modificar éste índice a lo largo del tiempo, se suele fijar el valor de frecuencia f_m (por tener otras implicaciones), variando la amplitud A_m.

2.3.1.1. Control del espectro en la síntesis FM

La ecuación característica de la síntesis FM para el caso en el que tenemos dos osciladores de tipo sinusoidal puede expresarse de la siguiente forma:

$$s(t) = A_c \cdot \text{sen}\,[2\pi f_c t + I\,\text{sen}\,(2\pi f_m t)] \tag{2.21}$$

Echando un vistazo a la expresión anterior, podemos fijarnos en que si la amplitud de la moduladora es nula ($A_m = 0$), el índice de modulación también será nulo ($I = 0$), por lo que no existirá modulación alguna, quedando la señal de salida según sigue: $s(t) = A_c \cdot \text{sen}\,(2\pi f_c t)$.

Ahora bien, para poder analizar el espectro de ésta señal de salida, debemos antes convertir su expresión característica a otra, de tal modo que

nos facilite ver el contenido en frecuencias que tendrá. Para ello, haremos uso de la siguiente identidad trigonométrica:

$$\operatorname{sen}(\theta + a \cdot \operatorname{sen}\beta) = \sum_{k=-\infty}^{+\infty} J_k(a) \cdot \operatorname{sen}(\theta + k\beta) \tag{2.22}$$

Donde las $J_k(a)$ se denominan *funciones de Bessel de primera especie*, teniendo en cuenta que para cada k existirá una función diferente que tomará valores según el parámetro a. Éste tipo de funciones son parecidas a senos amortiguados y más adelante volveremos a ellas.

Si identificamos los parámetros del primer miembro de ésta identidad con los parámetros de la fórmula FM (expresión 2.21), tendríamos que:

$$\begin{aligned} \theta &= 2\pi f_c t \\ a &= I \\ \beta &= 2\pi f_m t \end{aligned} \tag{2.23}$$

Por tanto, podemos reescribir la ecuación FM como:

$$s(t) = A_c \sum_{k=-\infty}^{+\infty} J_k(I) \cdot \operatorname{sen}(2\pi f_c t + k \cdot 2\pi f_m t) \tag{2.24}$$

En este punto, ya disponemos de una ecuación que nos proporciona información sobre el espectro de la señal.

Si analizamos la ecuación, vemos que se generan infinitos parciales alrededor de la frecuencia de la portadora, que se encuentran separados entre sí una frecuencia igual a la de la moduladora: $f_c + kf_m$, siendo $k = 0, \pm 1, \pm 2, \pm 3, ..., \pm\infty$. No obstante, aunque aparezcan infinitos parciales, cabe destacar que en la práctica solo se escucharán algunos, dado que las amplitudes de los mismos $J_k(I)$ irán disminuyendo a medida que k aumente, hasta que se hagan imperceptibles al oído. En éste sentido, la cantidad de parciales dependerá del índice de modulación (I).

Además, existe una fórmula empírica que nos permite obtener el número de parciales perceptibles, considerando como perceptibles aquellos cuya amplitud es mayor del 10 % (−20 dB) de la amplitud de pico de la señal. Así pues, el parcial de mayor orden K será:

$$K = I + 1 \tag{2.25}$$

Por tanto, el número total de parciales P de la señal modulada será:

$$P = 2K + 1 \tag{2.26}$$

Como conclusión, podemos considerar que los parciales generados serán $|f_c + kf_m|$, donde $k = 0, \pm1, \pm2, ..., \pm K$ y estarán separados entre sí el valor de la frecuencia de modulación (f_m). Como el parcial de mayor orden depende del índice de modulación I, éste único parámetro controlará la riqueza espectral del sonido sintetizado.

Veamos el siguiente ejemplo para ilustrar el procedimiento de cálculo del espectro generado.

Ejemplo: Obtener las frecuencias de los parciales generados en la señal de salida cuando realizamos una modulación FM entre dos osciladores sinusoidales, con $f_c = 1$ kHz, $f_m = 200$ Hz y un índice de modulación $I = 1$.

En primer lugar, calculamos el parcial de mayor orden perceptible, así como el número total de parciales perceptibles de la señal modulada:

$$K = I + 1 = 1 + 1 = 2 \tag{2.27}$$

$$P = 2K + 1 = 2 \cdot 2 + 1 = 5 \text{ parciales} \tag{2.28}$$

Por tanto, el parámetro k tomará los siguientes valores:

$$k = 0, \pm1, \pm2 \tag{2.29}$$

En segundo y último lugar, determinamos las frecuencias de los parciales mediante la expresión $|f_c + kf_m|$ para los valores de $f_c = 1000$ Hz y $f_m = 200$ Hz, agrupando los datos en una tabla:

Parcial (k)	...	-2	-1	0	1	2	...
Frecuencia	...	$\|f_c - 2f_m\|$	$\|f_c - f_m\|$	$\|f_c\|$	$\|f_c + f_m\|$	$\|f_c + 2f_m\|$	...
Frecuencia (Hz)	...	600	800	1000	1200	1400	...

Tabla 2.1: Frecuencias de los parciales en una modulación FM con 2 osciladores sinusoidales para $f_c = 1$ kHz, $f_m = 200$ Hz e $I = 1$.

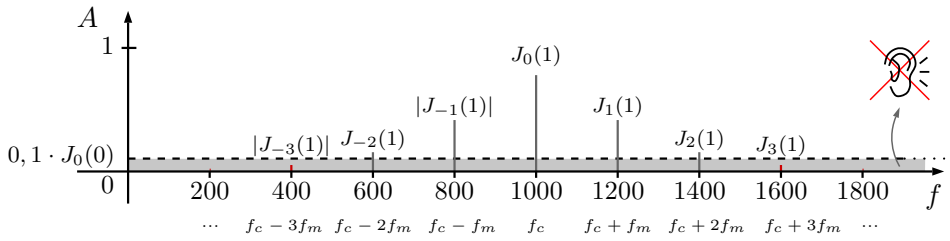


Figura 2.29: Representación del espectro de una modulación FM con 2 osciladores sinusoidales para $f_c = 1$ kHz, $f_m = 200$ Hz e $I = 1$.

En la Fig. 2.29, podemos observar la representación del espectro obtenido para éste ejemplo. Nótese cómo todas las frecuencias perceptibles son múltiplos (armónicos) de la frecuencia de modulación $f_m = 200$ Hz. Además de ello, también puede observarse que por encima de $k = 2$ y por debajo de $k = -2$, dejan de percibirse los parciales sucesivos, por presentar una amplitud inferior al 10 % de la amplitud pico de la señal, que en este caso sería $J_0(1)$. Finalmente, cada parcial presenta la amplitud correspondiente al valor de la función de *Bessel* para el índice de modulación $I = 1$ ($J_k(1)$). Si nos fijamos en los parciales de índices $k = -1$ y $k = -3$, las amplitudes $J_{-1}(1)$ y $J_{-3}(1)$ se han puesto en valor absoluto para que todas queden positivas; no obstante, más adelante veremos que en realidad ambas son negativas.

Para finalizar éste apartado, realizaremos otro ejemplo, solo que en ésta ocasión veremos la importancia de la variación del índice de modulación a la hora de generar nuevos parciales.

Ejemplo: Considerando las mismas frecuencias de portadora y moduladora que en el ejemplo anterior, determinar las frecuencias de los parciales cuando el índice de modulación toma los siguientes valores: $I = 0, 1, 2, 3, 4$.

Se trata en éste caso de llevar a cabo el mismo procedimiento de cálculo para cada índice de modulación, recogiendo de nuevo los valores en una tabla:

Parcial (k)	-5	-4	-3	-2	-1	0	1	2	3	4	5
f **(Hz)** $I = 0$						1000					
f **(Hz)** $I = 1$				600	800	1000	1200	1400			
f **(Hz)** $I = 2$			400	600	800	1000	1200	1400	1600		
f **(Hz)** $I = 3$		200	400	600	800	1000	1200	1400	1600	1800	
f **(Hz)** $I = 4$	0	200	400	600	800	1000	1200	1400	1600	1800	2000

Tabla 2.2: Frecuencias de los parciales en una modulación FM con 2 osciladores sinusoidales para $f_c = 1$ kHz, $f_m = 200$ Hz e I variable.

Resulta necesario destacar que cuando el índice de modulación es nulo ($I = 0$) no existirá modulación, por lo que se establece el parcial de mayor orden a cero ($K = 0$) y por consiguiente solo existirá un parcial en la señal de salida (el de la portadora)[5].

En la Fig. 2.30, se representan los distintos parciales para cada valor del índice de modulación I. Como vemos, el número de parciales alrededor de la portadora crece rápidamente conforme aumentamos el valor del parámetro I. A diferencia del ejemplo anterior, hemos considerado las amplitudes reales de los parciales, por lo que algunas son negativas debido a las propiedades de las funciones de *Bessel* $J_k(I)$.

[5]En la expresión $K = I + 1$, en caso de que el índice de modulación I sea decimal, se aproximaría al entero más cercano.

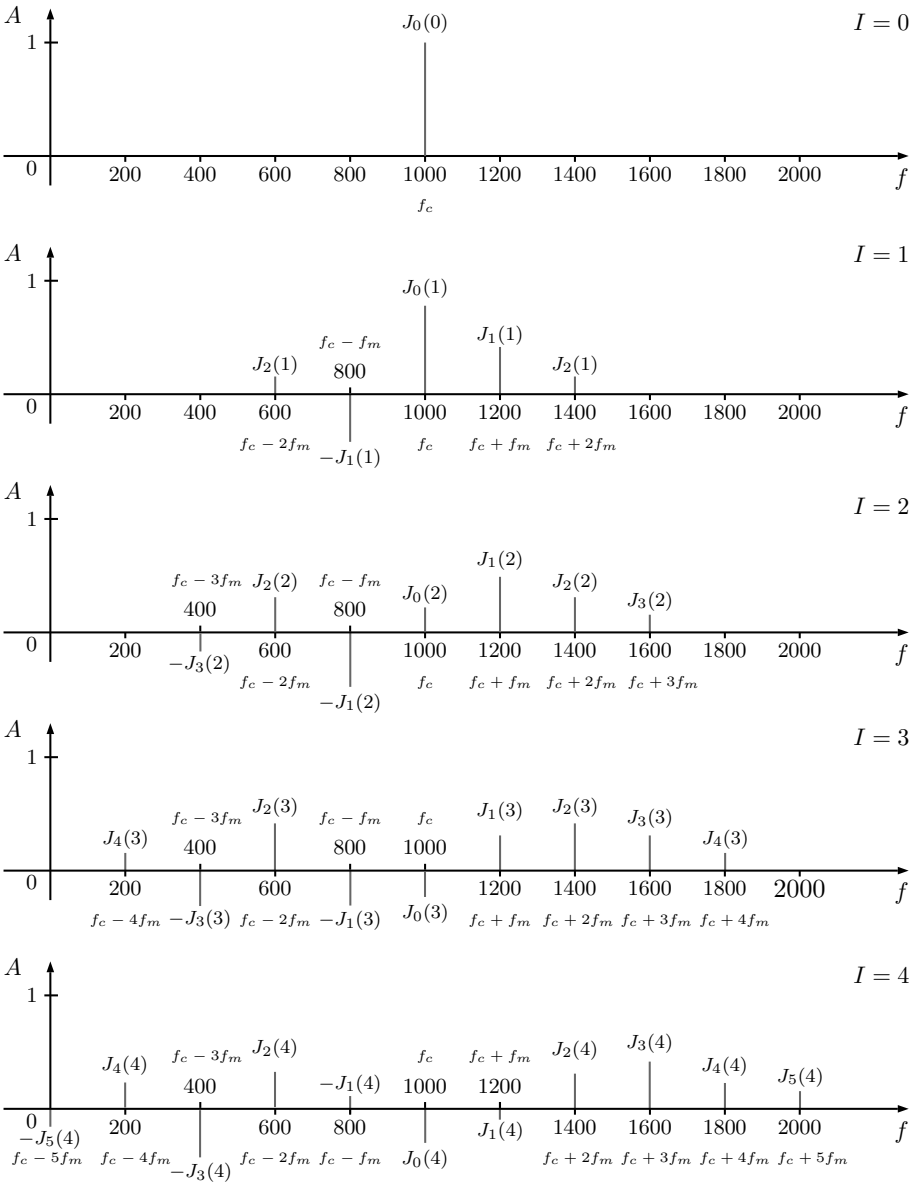


Figura 2.30: Representación del espectro obtenido para cada valor de índice de modulación en una modulación FM con 2 osciladores para $f_c = 1$ kHz y $f_m = 200$ Hz.

2.3.1.2. Control de las amplitudes del espectro FM

Ya sabemos cómo estarán distribuidos los parciales en una modulación FM simple y que las amplitudes de los mismos estarán controladas por las *funciones de Bessel* de primera especie $J_k(I)$, donde $k = 0, \pm1, \pm2, \pm3, ..., \pm K$.

También hemos podido comprobar que cuando $k = 0$, tenemos la función de *Bessel* $J_0(I)$, que coincide con el valor de amplitud que tendrá la portadora, siguiendo el resto de parciales una distribución simétrica con respecto a ésta.

En cualquier caso, los valores de dichas funciones vendrán dados por el índice de modulación I. A medida que aumentemos el valor de I, no solo aparecerán nuevos parciales alrededor de la portadora, sino que además dichas bandas laterales restarán amplitud a la portadora, debido a que la energía total se reparte entre todos los parciales. Ésto significa que podemos modificar el timbre de la señal de salida sin que varíe la amplitud global del sonido.

Otro aspecto a tener en cuenta es la función de las amplitudes de la portadora y moduladora (A_c y A_m, respectivamente). Pudimos observar que cuando la amplitud de la moduladora es nula ($A_m = 0$), no existirá modulación, por lo que el único parcial del espectro de la señal de salida será el de la portadora. Por otra parte, cuando la amplitud de la portadora es nula ($A_c = 0$), no se oirá nada porque la señal de salida será nula. Por tanto, la amplitud de la moduladora A_m controla el índice de modulación (al estar f_m fijada), por lo que modificará el timbre de la señal de salida, mientras que la amplitud de la portadora (A_c) controlará la amplitud global de la señal final.

Una característica intrínseca de la síntesis FM es que las amplitudes relativas de los parciales no pueden controlarse de manera individual, puesto que están gobernadas por los valores de las funciones $J_k(I)$.

Las funciones de *Bessel* de primera especie pueden calcularse a partir de la siguiente serie de potencias:

$$J_k(I) = \sum_{n=0}^{\infty} \frac{(-1)^n}{n!\,(n+k)!} \cdot \left(\frac{I}{2}\right)^{2n+k} \tag{2.30}$$

Para proceder al cálculo de las $J_k(I)$, truncamos el límite superior del sumatorio a un número de elementos suficientemente grande para que tengamos una buena aproximación (por ejemplo, a 50 elementos).

No obstante, éstas funciones las podemos encontrar tabuladas en la bibliografía. En la tabla 2.3, se han recogido los valores de las 11 primeras funciones de *Bessel* para un rango de índice de modulación comprendido entre 0 y 10, siendo más precisos entre 0 y 1. Como vemos, no se han incluido funciones de *Bessel* con índice negativo ($k < 0$), que se corresponderían con las amplitudes de los parciales que se encuentran a la izquierda de la portadora. Ésto es debido a que existe una propiedad que nos dice lo siguiente:

$$J_{-k}(I) = (-1)^k \cdot J_k(I) \tag{2.31}$$

I	J_0	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9	J_{10}
0,0	1, 000	0, 000	0, 000	0, 000	0, 000	0, 000	0, 000	0, 000	0, 000	0, 000	0, 000
0,25	0, 984	0, 124	0, 008	0, 000	0, 000	0, 000	0, 000	0, 000	0, 000	0, 000	0, 000
0,5	0, 938	0, 242	0, 031	0, 003	0, 000	0, 000	0, 000	0, 000	0, 000	0, 000	0, 000
0,75	0, 864	0, 349	0, 067	0, 008	0, 000	0, 000	0, 000	0, 000	0, 000	0, 000	0, 000
1,0	0, 765	0, 440	0, 115	0, 020	0, 002	0, 000	0, 000	0, 000	0, 000	0, 000	0, 000
1,5	0, 512	0, 558	0, 232	0, 061	0, 012	0, 002	0, 000	0, 000	0, 000	0, 000	0, 000
2,0	0, 224	0, 577	0, 353	0, 129	0, 034	0, 007	0, 001	0, 000	0, 000	0, 000	0, 000
2,5	-0,048	0, 497	0, 446	0, 217	0, 074	0, 020	0, 004	0, 001	0, 000	0, 000	0, 000
3,0	-0,260	0, 339	0, 486	0, 309	0, 132	0, 043	0, 011	0, 003	0, 000	0, 000	0, 000
3,5	-0,380	0, 137	0, 459	0, 387	0, 204	0, 080	0, 020	0, 007	0, 002	0, 000	0, 000
4,0	-0,397	-0,066	0, 364	0, 430	0, 281	0, 132	0, 049	0, 015	0, 004	0, 001	0, 000
4,5	-0,321	-0,231	0, 218	0, 425	0, 348	0, 195	0, 084	0, 030	0, 009	0, 002	0, 001
5,0	-0,178	-0,328	0, 047	0, 365	0, 391	0, 261	0, 131	0, 053	0, 018	0, 006	0, 001
5,5	-0,007	-0,341	-0,117	0, 256	0, 397	0, 321	0, 187	0, 087	0, 034	0, 011	0, 003
6,0	0, 151	-0,277	-0,243	0, 115	0, 358	0, 362	0, 246	0, 130	0, 057	0, 021	0, 007
6,5	0, 260	-0,154	-0,307	-0,035	0, 275	0, 374	0, 300	0, 180	0, 088	0, 037	0, 013
7,0	0, 300	-0,005	-0,301	-0,168	0, 158	0, 348	0, 339	0, 234	0, 128	0, 059	0, 025
7,5	0, 266	0, 135	-0,230	-0,258	0, 024	0, 283	0, 354	0, 283	0, 174	0, 089	0, 039
8,0	0, 172	0, 235	-0,113	-0,291	-0,105	0, 186	0, 338	0, 321	0, 223	0, 126	0, 061
9,0	-0,090	0, 245	0, 145	-0,181	-0,265	-0,055	0, 204	0, 327	0, 305	0, 215	0, 125
10,0	-0,246	0, 043	0, 255	0, 058	-0,220	-0,234	-0,014	0, 217	0, 318	0, 292	0, 207

Tabla 2.3: Funciones de *Bessel* de primera especie para valores de índice de modulación entre 0 y 10.

Ésto quiere decir que el valor de una función de *Bessel* con índice negativo siempre se puede expresar en función del valor de su opuesta (función con el mismo índice pero positivo), de tal manera que si el índice k es par, las amplitudes de los parciales de la banda izquierda serán idénticos a los de la banda derecha, mientras que si k es impar, las amplitudes de los parciales de la banda izquierda serán como los de la derecha pero con signo cambiado. Éste hecho se refleja en la Fig. 2.30.

La representación de las seis primeras funciones de *Bessel* en función del índice de modulación se presenta en la siguiente figura:

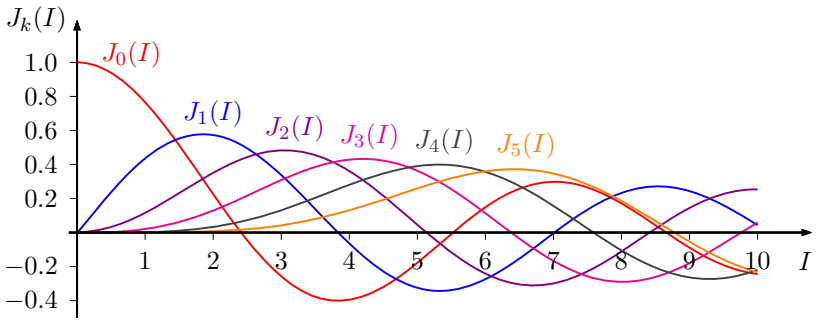


Figura 2.31: Representación de diversas funciones de *Bessel* de primera especie.

Para más información sobre las funciones de *Bessel*, consulte [7]. Prosigamos con el siguiente ejemplo para poner en práctica lo estudiado.

Ejemplo: Determínese la amplitud y frecuencia de los parciales generados en una modulación FM simple de dos osciladores sinusoidales para los

parámetros $A_c = 0,75$, $f_c = 800$ Hz, $A_m = 200$ Hz y $f_m = 100$ Hz.

En primer lugar, calculamos el índice de modulación a partir de la amplitud y frecuencia de la señal moduladora:

$$I = \frac{A_m}{f_m} = \frac{200\,\text{Hz}}{100\,\text{Hz}} = 2 \tag{2.32}$$

Continuamos calculando el parcial de mayor orden, así como el número total de parciales del espectro (perceptibles) de la señal de salida:

$$K = I + 1 = 2 + 1 = 3 \tag{2.33}$$

$$P = 2K + 1 = 2 \cdot 3 + 1 = 7 \text{ parciales} \tag{2.34}$$

Por tanto, el índice k tomará los siguientes valores:

$$k = 0, \pm 1, \pm 2, \pm 3 \tag{2.35}$$

Para obtener las frecuencias de los parciales, aplicamos la conocida fórmula $|f_c + kf_m|$, que particularizada a los datos del ejemplo quedaría $|800 + k \cdot 100|$, sustituyendo k por el índice del parcial correspondiente.

Ahora bien, para calcular las amplitudes, debemos tener en cuenta que al no tener una amplitud de portadora $A_c = 1$, los valores de amplitud de cada parcial no coincidirán exactamente con las funciones de *Bessel* $J_k(2)$, sino que habrá que multiplicarlas por el valor de la amplitud de la portadora (para éste ejemplo $A_c = 0,75$). Por consiguiente, la amplitud de cada parcial será: $A_c \cdot J_k(2) = 0,75 \cdot J_k(2)$.

Calculadas las frecuencias y amplitudes de los parciales, procedemos a colocarlas en una tabla:

Parcial (k)	-3	-2	-1	0	1	2	3
Frecuencia (Hz)	500	600	700	800	900	1000	1100
Amplitud	$-J_3(2)$	$J_2(2)$	$-J_1(2)$	$J_0(2)$	$J_1(2)$	$J_2(2)$	$J_3(2)$
($\times 0,75$)	-0,097	0,265	-0,433	0,168	0,433	0,265	0,097

Tabla 2.4: Frecuencias y amplitudes de los parciales del espectro generado para una modulación FM simple con los datos del ejemplo.

Recordemos que las amplitudes negativas indican que el parcial presenta inversión de fase ($\pm 180^o$), que es equivalente a que el parcial tenga frecuencia negativa: $\text{sen}\,(f \pm 180^o) = \text{sen}\,(-f) = -\text{sen}\,(f)$.

Ésto es especialmente útil en el caso de interferencias constructivas o destructivas cuando se superponen dos o más parciales de misma frecuencia, de modo que si están en fase sus amplitudes se sumarán, mientras que si están en contrafase las amplitudes se restarán.

2.3.1.3. Control de la armonicidad del espectro FM

Resulta fundamental conocer la armonicidad del espectro generado cuando realizamos síntesis FM, puesto que de ello dependerá que escuchemos el sonido "afinado" en mayor o menor medida. La armonicidad del espectro depende de las posiciones relativas que haya entre los parciales. Dichas posiciones dependen, a su vez, de la relación entre las frecuencias de las señales portadora y moduladora, es decir:

$$\frac{f_c}{f_m} \tag{2.36}$$

A la hora de trabajar con ésta fracción, es necesario simplificarla para obtener la fracción reducible correspondiente (de numerador y denominador enteros si es posible) que se expresa como $c : m$. Dependiendo del valor de ésta fracción, averiguaremos la armonicidad del espectro:

- Si $c : m$ es un número *entero*: El espectro generado será *armónico*.
- Si $c : m$ es un número *racional*: El espectro generado será *menos armónico*. Dependiendo de lo grandes que sean los enteros c y m obtendremos espectros armónicos a los que les faltan algunos parciales, o bien espectros armónicos con la frecuencia fundamental f_0 inaudible (por ser muy baja), en cuyo caso se oirán como inarmónicos.
- Si $c : m$ es un número *irracional*: El espectro generado será totalmente *inarmónico*.

Para poder determinar la *frecuencia fundamental* f_0 (audible o no) en cualquiera de los dos primeros casos, podemos emplear la ecuación que se muestra a continuación:

$$f_0 = \frac{f_c}{c} = \frac{f_m}{m} \tag{2.37}$$

Pudiendo emplear en el cálculo cualquiera de las dos fracciones indistintamente.

Analicemos un ejemplo sencillo para comprobar la armonicidad del espectro en varios casos.

Ejemplo: Obtenga la relación $c : m$, así como el espectro completo en una modulación FM simple con dos osciladores sinusoidales para los siguientes valores de amplitud y frecuencia de portadora y moduladora:

a) $A_c = 6000$, $f_c = 1000$ Hz, $A_m = 1000$ Hz y $f_m = 500$ Hz.

b) $A_c = 500$, $f_c = 800$ Hz, $A_m = 405$ Hz y $f_m = 405$ Hz.

c) $A_c = 1000$, $f_c = 1,2$ kHz, $A_m = 520$ Hz y $f_m = 692,8$ Hz.

Para resolver el apartado *a*), comenzamos obteniendo la relación $c : m$ a partir de las frecuencias dadas:

$$\frac{f_c}{f_m} = \frac{1000 \text{ Hz}}{500 \text{ Hz}} = 2 \tag{2.38}$$

$$c : m = 2 : 1 = 2 \tag{2.39}$$

Por tanto, podemos asegurar que dado que $c : m$ es un número entero, el espectro será armónico y la frecuencia fundamental será:

$$f_0 = \frac{f_c}{c} = \frac{f_m}{m} = \frac{1000 \text{ Hz}}{2} = \frac{500 \text{ Hz}}{1} = 500 \text{ Hz} \tag{2.40}$$

Ahora obtenemos el índice de modulación, el parcial de mayor orden y el número total de parciales:

$$I = \frac{A_m}{f_m} = \frac{1000 \text{ Hz}}{500 \text{ Hz}} = 2 \tag{2.41}$$

$$K = I + 1 = 2 + 1 = 3 \tag{2.42}$$

$$P = 2K + 1 = 6 + 1 = 7 \text{ parciales} \tag{2.43}$$

Ahora ya podemos obtener las amplitudes y frecuencias de los parciales del espectro:

Parcial (k)	-3	-2	-1	0	1	2	3
Frecuencia (Hz)	-500	0	500	1000	1500	2000	2500
Amplitud	$-J_3(2)$	$J_2(2)$	$-J_1(2)$	$J_0(2)$	$J_1(2)$	$J_2(2)$	$J_3(2)$
(×6000)	(+)774	2118	-3462	1344	3462	2118	774

Tabla 2.5: Frecuencias y amplitudes de los parciales del espectro generado para el apartado *a*) (espectro armónico).

Observemos que aparece una frecuencia negativa para $k = -3$ (-500 Hz). Esto quiere decir (como se comentó al final del apartado anterior) que se producirá un cambio de signo en la amplitud final por la alternancia de fase provocada por ésta frecuencia negativa. Así pues, toda frecuencia negativa puede convertirse a positiva cambiando el signo de la amplitud final.

Si nos fijamos en la tabla anterior, podemos apreciar que para -500 Hz, la amplitud final será $-6000 \cdot J_3(2)$, por tratarse de un parcial con índice negativo e impar ($k = -3$), pero pasando la frecuencia a positiva (500 Hz) cambiamos el signo final de la amplitud final (se ha enfatizado éste hecho colocando entre paréntesis el signo final +).

Nótese que se produciría una interferencia destructiva para la frecuencia de 500 Hz, puesto que se restarían las amplitudes de los parciales de índice $k = -3$ y $k = -1$, por presentar ambos la misma frecuencia. Por consiguiente, la amplitud final para el parcial de 500 Hz sería $774 - 3462 = -2688$.

Por tanto, el espectro generado para el apartado *a*) sería el que se muestra en la figura adjunta:

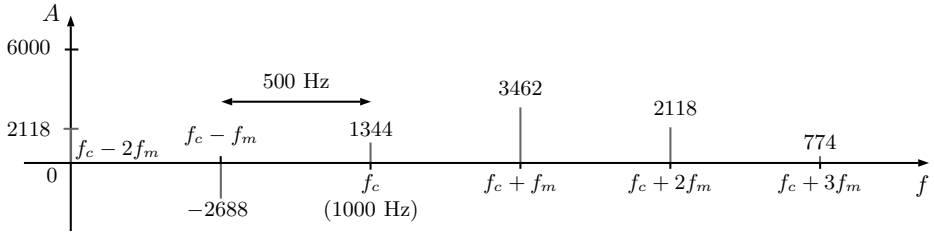


Figura 2.32: Espectro generado para el apartado *a*) (armónico).

Para solucionar el apartado *b*), procedemos de la misma forma, calculando primero la relación $c : m$:

$$\frac{f_c}{f_m} = \frac{800 \text{ Hz}}{405 \text{ Hz}} = \frac{160}{81} \tag{2.44}$$

$$c : m = 160 : 81 \tag{2.45}$$

Al ser la relación $c : m$ un número racional, el espectro será inarmónico en la práctica ya que la frecuencia fundamental no es audible (por ser < 20 Hz):

$$f_0 = \frac{f_c}{c} = \frac{f_m}{m} = \frac{800 \text{ Hz}}{160} = \frac{405 \text{ Hz}}{81} = 5 \text{ Hz} \tag{2.46}$$

Determinemos ahora el índice de modulación, el parcial de mayor orden y el número total de parciales:

$$I = \frac{A_m}{f_m} = \frac{405 \text{ Hz}}{405 \text{ Hz}} = 1 \tag{2.47}$$

$$K = I + 1 = 1 + 1 = 2 \tag{2.48}$$

$$P = 2K + 1 = 4 + 1 = 5 \text{ parciales} \tag{2.49}$$

Obtengamos las frecuencias y amplitudes de los parciales del espectro generado, así como su representación:

Parcial (k)	-2	-1	0	1	2
Frecuencia (Hz)	-10	395	800	1205	1610
Amplitud	$J_2(1)$	$-J_1(1)$	$J_0(1)$	$J_1(1)$	$J_2(1)$
(×500)	(-)57,5	-220	382, 5	220	57, 5

Tabla 2.6: Frecuencias y amplitudes de los parciales del espectro generado para el apartado *b*) (espectro inarmónico por f_0 inaudible).

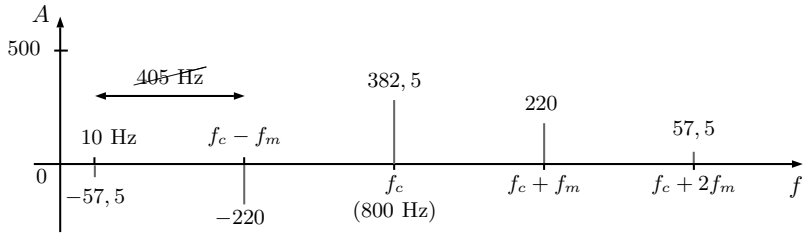


Figura 2.33: Espectro generado para el apartado *b*) (inarmónico por tener f_0 inaudible).

Por último, pasamos a resolver el apartado *c*), realizando el mismo procedimiento que en los dos apartados anteriores:

$$\frac{f_c}{f_m} = \frac{1200 \text{ Hz}}{692,8 \text{ Hz}} = \frac{750}{433} \tag{2.50}$$

$$c : m \simeq \sqrt{3} \tag{2.51}$$

Éste valor de relación $c : m$ nos está indicando que el espectro será totalmente inarmónico y, consecuentemente, no existirá frecuencia fundamental f_0.

Calculamos el índice de modulación, el parcial de mayor orden y el número total de parciales:

$$I = \frac{A_m}{f_m} = \frac{520 \text{ Hz}}{692,8 \text{ Hz}} \simeq 0,75 \tag{2.52}$$

En éste caso, consideramos $K = 2$ puesto que la tabla 2.3 presenta solo 2 amplitudes para $J_k(0,75)$ ($0,864$ y $0,349$) que se encuentran por encima del $10\,\%$ de la mayor amplitud ($> 0,1$):

$$K = 2 \tag{2.53}$$

$$P = 2K + 1 = 4 + 1 = 5 \text{ parciales} \tag{2.54}$$

Por último, determinamos los valores de amplitudes y frecuencia del espectro y procedemos a su representación gráfica:

Parcial (k)	-2	-1	0	1	2
Frecuencia (Hz)	-185,6	$507,2$	1200	$1892,8$	$2585,6$
Amplitud	$J_2(0,75)$	$-J_1(0,75)$	$J_0(0,75)$	$J_1(0,75)$	$J_2(0,75)$
(×1000)	(-)67	-349	864	349	67

Tabla 2.7: Frecuencias y amplitudes de los parciales del espectro generado para el apartado *c*) (espectro completamente inarmónico).

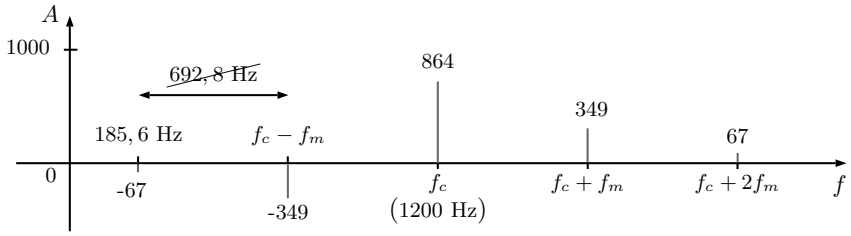


Figura 2.34: Espectro generado para el apartado *c*) (completamente inarmónico).

2.3.1.4. Control dinámico del espectro FM

Hasta ahora, hemos conseguido sintetizar señales con espectros constantes debido a que hemos trabajado con índices de modulación también constantes.

Cuando hacemos que el índice de modulación varíe a lo largo del tiempo, hablamos de envolvente temporal del índice de modulación y la representamos como $I(t)$. Con ésta envolvente lograremos que el espectro de la señal generada sea dinámico, creándose o eliminándose parciales cuya amplitud variará dependiendo del valor de las funciones de *Bessel* para el índice de modulación que se tenga en un instante concreto.

En éste sentido, también se puede crear una envolvente para la relación $c : m$, de modo que la variación de la curva permita que el espectro de la señal se modifique de manera continua entre armónico e inarmónico.

Otro tipo de envolventes que pueden ayudar a que el espectro se modifique en función del tiempo serían las envolventes aplicadas a la amplitud de la portadora A_c.

En cualquier caso y a la hora de la implementación, todas las curvas mencionadas se almacenan mediante una tabla de valores (como en el caso de los osciladores, por ejemplo) y se suelen dar normalizadas (el valor máximo al que puede llegar la señal es 1).

A continuación pueden verse un par de ejemplos de envolventes diseñadas para I y A_c a la hora de formar dos tipos de instrumentos: un instrumento de tipo metal y otro de tipo percusión (Fig. 2.35).

En el instrumento de metal podemos apreciar que tanto la envolvente de I como la de A_c son idénticas (de tipo ADSR), con lo que el número de parciales generados crecerá o decrecerá tan rápido como aumente o dismi-

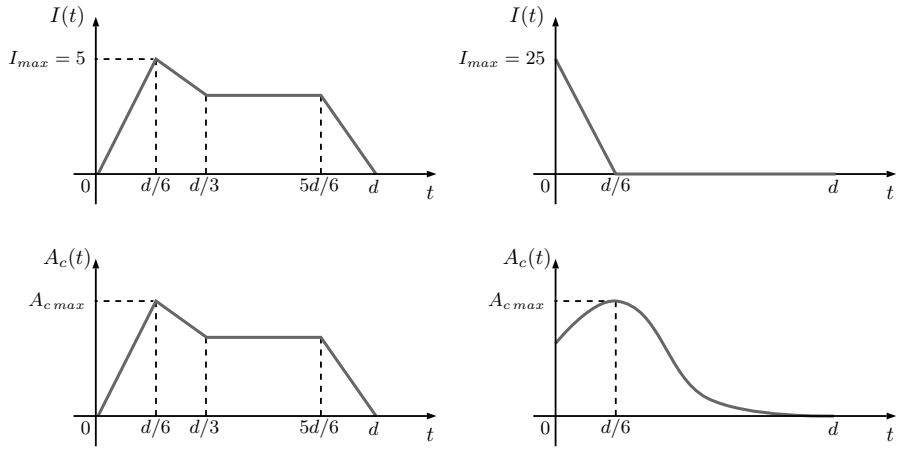


Figura 2.35: Envolventes del índice de modulación (I) y amplitud de portadora (A_c) para el diseño de dos instrumentos FM: de tipo metal (izda) y de tipo percusión (dcha). Ambos ejemplos han sido extraídos de [1].

nuya la amplitud global de la señal, respectivamente. En el instrumento de percusión, podemos observar que gracias a la envolvente de I (tipo AR) se generan muchos armónicos en el instante $t = 0$ (por el gran valor que toma I), y a medida que sucede la fase de relajación, comienza a descender rápidamente el número de parciales hasta llegar al instante $d/6$. La envolvente de amplitud de A_c crece hasta alcanzar el instante $d/6$, y a partir de aquí desciende de forma similar a una exponencial negativa hasta llegar al instante d[6].

2.3.2. Síntesis FM compuesta

La síntesis FM simple presenta el inconveniente de que no podemos controlar de manera individual la amplitud de los parciales generados del espectro de la señal. Además de ello, el hecho de que exista un control tan rígido debido a las funciones de *Bessel* $J_k(I)$, hace que los timbres sintetizados mediante FM simple tiendan a parecerse.

La solución a éste problema reside en la síntesis FM compuesta, de modo que se pueden emplear varias señales portadoras y/o moduladoras complejas a través distintos tipos de asociaciones entre osciladores. Aplicando ésto, se consigue un resultado en el que cada parcial se comportará como un oscilador sinusoidal independiente.

Los osciladores que intervienen en dichas asociaciones para sintetizar

[6]Normalmente, la duración d suele ser proporcional a la duración de la nota, aunque no tiene por qué ser así.

la señal de salida, pueden producir señales que presenten más de un parcial (por ejemplo, un oscilador que genere una señal cuadrada), pero por contra, los parciales del espectro de ésta señal también evolucionarán de forma conjunta. Ésto puede suponer una limitación en cuanto a la libertad del diseño sonoro. No obstante, ésta limitación se puede paliar si utilizamos un número N de osciladores de tipo sinusoidal (donde $N > 2$), de modo que se permitan definir envolventes de control individuales para cada componente.

Ya sabemos que mediante FM simple podemos lograr espectros muy ricos en parciales, haciendo uso solo de dos osciladores. En FM compuesta no se suele utilizar más de 3 o 4 moduladoras/portadoras sinusoidales debido al gran efecto multiplicador que supondría trabajar con un número elevado de osciladores.

Cuando hablamos de síntesis FM compuesta, se frecuenta el uso de dos términos ya vistos en el anterior capítulo[7]: *operadores* y *algoritmos*. Recordemos que los operadores hacen referencia a los osciladores, mientras que los algoritmos son las distintas formas de interconexión entre los osciladores.

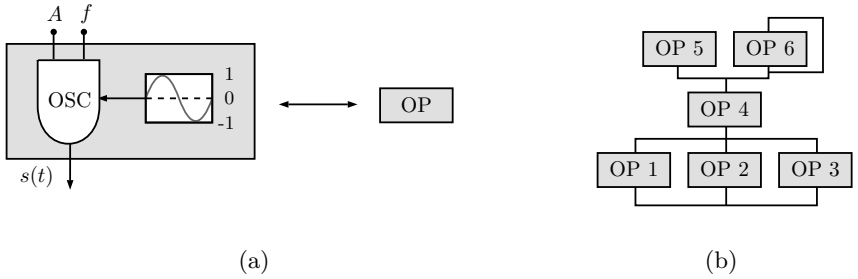


Figura 2.36: (a) Diagrama detallado y simplificado de un operador. (b) Ejemplo de un algoritmo FM de 6 operadores.

A continuación, se analizarán cuatro asociaciones o algoritmos fundamentales en lo que concierne a la síntesis FM compleja.

2.3.2.1. Modulación FM en serie

La modulación FM en serie se produce cuando una única señal modula a diversas señales portadoras. El resultado de ésta asociación equivaldría a sumar cada señal FM simple generada por cada portadora, teniendo en cuenta que la modulación será la misma para todas (Fig. 2.37). La ecuación que nos describe una modulación FM en serie para N portadoras la podemos

[7] Véase Fig. 1.3.

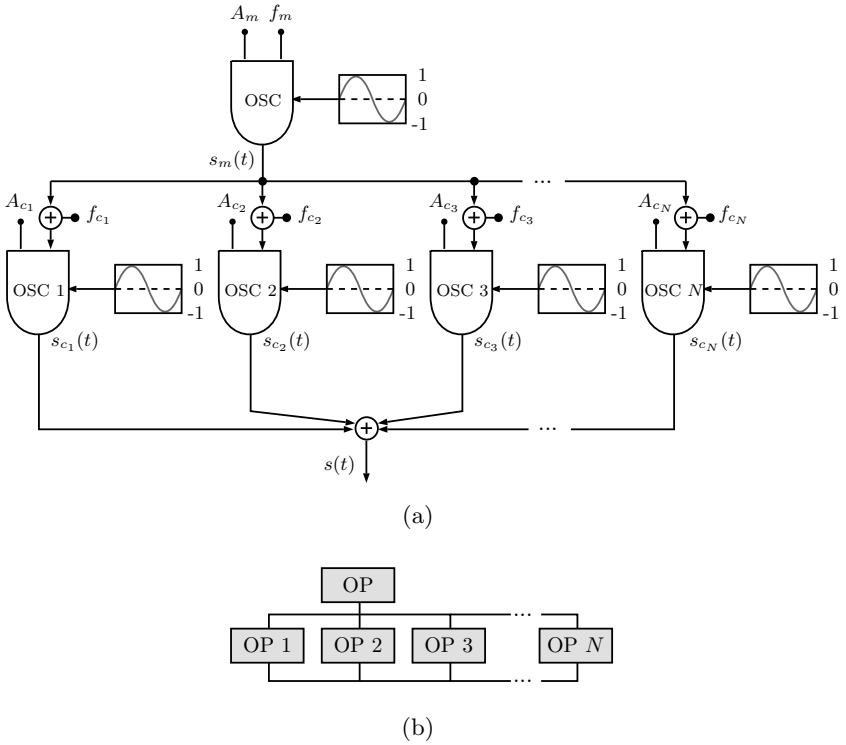


Figura 2.37: Diagrama detallado (a) y simplificado (b) del algoritmo de modulación FM serie para N osciladores portadores.

obtener a partir de la ecuación de la FM simple:

$$s(t) = \sum_{n=1}^{N} \left(A_{c_n} \sum_{k=-\infty}^{+\infty} J_k(I) \cdot \text{sen}(2\pi f_{c_n} t + k \cdot 2\pi f_m t) \right) \tag{2.55}$$

Puesto que solo existe una moduladora, el índice de modulación será el mismo para todas las portadoras y valdrá lo mismo que en FM simple:

$$I = \frac{A_m}{f_m} \tag{2.56}$$

Como cada portadora generará $P_n = 2K + 1$ parciales y hay N portadoras, el número total de parciales generados será:

$$P = N \cdot P_n = N \cdot (2K + 1) \tag{2.57}$$

Por otro lado, al número total de parciales P habría que restar el número de posibles superposiciones. Se producen superposiciones entre parciales cuando la diferencia de frecuencias entre dos de las portadoras es un múltiplo entero de la frecuencia de modulación.

Por ejemplo, si suponemos dos frecuencias de portadora f_{c_1} y f_{c_2} (donde $f_{c_2} > f_{c_1}$), cumpliéndose que $f_{c_2} - f_{c_1} = nf_m$, se puede asegurar que si n es par, existirá un valor de $k = n/2$ para el cual los parciales de frecuencias $f_{c_1} + kf_m$ y $f_{c_2} - kf_m$ coincidirán. En caso de que n fuera impar, se producirían también superposiciones para distintos valores de k de ambas portadoras: $k_1 = n - k_2$.

En cualquiera de éstos casos, al coincidir las frecuencias de algunos parciales de cada portadora, habría que sumar las amplitudes de los mismos, con sus respectivos signos dependiendo de las fases de cada parcial.

Estudiemos, a continuación, el siguiente ejemplo de síntesis FM serie:

Ejemplo: Obtenga el espectro completo para una modulación FM compuesta tipo serie con una moduladora y tres portadoras, estudiando las posibles superposiciones de parciales. Datos de las señales: $A_m = 400$ Hz, $f_m = 200$ Hz, $A_{c_1} = 600$, $f_{c_1} = 400$ Hz, $A_{c_2} = 800$, $f_{c_2} = 1000$ Hz, $A_{c_3} = 1000$ y $f_{c_3} = 1200$ Hz.

Como se trata de una modulación FM en serie, únicamente habrá una moduladora, por lo que el índice de modulación será común para todas las portadoras:

$$I = \frac{A_m}{f_m} = \frac{400 \text{ Hz}}{200 \text{ Hz}} = 2 \tag{2.58}$$

El parcial de mayor orden y el número de parciales por portadora será también idéntico para todas ellas:

$$K = I + 1 = 2 + 1 = 3 \tag{2.59}$$

$$P_n = 2K + 1 = 6 + 1 = 7 \text{ parciales/portadora} \tag{2.60}$$

Como hay $N = 3$ portadoras, el número total de parciales generados será:

$$P = N \cdot P_n = N \cdot (2K + 1) = 3 \cdot 7 = 21 \text{ parciales} \tag{2.61}$$

A continuación, obtenemos la relación $c : m$ para estudiar la armonicidad de cada portadora con la moduladora:

$$\begin{aligned} c_1 : m &= 2 : 1 \\ c_2 : m &= 5 : 1 \\ c_3 : m &= 6 : 1 \end{aligned} \tag{2.62}$$

En vista de los resultados obtenidos, podemos deducir que todos los parciales generados son armónicos porque las relaciones $c : m$ dan números enteros. La frecuencia fundamental tendrá por tanto el siguiente valor:

$$f_0 = \frac{f_{c_1}}{c_1} = \frac{f_{c_2}}{c_2} = \frac{f_{c_3}}{c_3} = \frac{f_m}{m} = \frac{200 \text{ Hz}}{1} = 200 \text{ Hz} \tag{2.63}$$

Calculemos las frecuencias y amplitudes de los parciales de cada portadora:

Parcial (k)	-3	-2	-1	0	1	2	3
fs_{c_1} (Hz)	-200	0	200	**400**	**600**	**800**	**1000**
fs_{c_2} (Hz)	**400**	**600**	**800**	**1000**	**1200**	**1400**	**1600**
fs_{c_3} (Hz)	**600**	**800**	**1000**	**1200**	**1400**	**1600**	1800
Amplitud	$-J_3(2)$	$J_2(2)$	$-J_1(2)$	$J_0(2)$	$J_1(2)$	$J_2(2)$	$J_3(2)$
As_{c_1} ($\times 600$)	-77,4	211,8	-346,2	134,4	346,2	211,8	77,4
As_{c_2} ($\times 800$)	-103,2	282,4	-461,6	179,2	461,6	282,4	103,2
As_{c_3} ($\times 1000$)	-129	353	-577	224	577	353	129

Tabla 2.8: Frecuencias y amplitudes de los parciales de las tres portadoras. Las superposiciones se indican mediante negrita y subrayado.

Como podemos ver, se producen superposiciones en los parciales generados, bien sea porque coinciden parciales cuyas frecuencias son positivas y negativas del mismo valor absoluto (subrayadas en la tabla); o bien porque coinciden directamente las frecuencias de parciales generados por distintas portadoras (señaladas en negrita).

En el caso de las frecuencias subrayadas, podemos apreciar que para la primera portadora (s_{c_1}), las frecuencias -200 Hz y 200 Hz ($k = -3$ y $k = -1$, respectivamente) difieren únicamente en la fase, por lo tanto, la frecuencia negativa tendrá que invertir su amplitud para convertirse en positiva. Entonces, la amplitud final para la frecuencia de 200 Hz será: $600 \cdot (-J_1(2) + J_3(2)) = -346,2 + 77,4 = -268,8$.

Para los casos en los que las frecuencias aparecen en negrita, habrá que sumar las amplitudes de los parciales que coincidan en frecuencia, teniendo en cuenta el correspondiente signo. Empecemos con el caso de la frecuencia de 400 Hz. Nótese que dicha frecuencia aparece para el parcial $k = 0$ de la portadora s_{c_1} y para el parcial $k = -3$ de la portadora s_{c_2}. Por tanto, la amplitud final para la frecuencia de 400 Hz será: $-800J_3(2) + 600J_0(2) = -103,2 + 134,4 = 31,2$.

Si tomamos por ejemplo la frecuencia de 600 Hz, se produce una superposición de los parciales $k = 1$ de s_{c_1}, $k = -2$ de s_{c_2} y $k = -3$ de s_{c_3}. Por consiguiente, habrá que agrupar las amplitudes de los parciales mencionados del siguiente modo: $600J_1(2)+800J_2(2)-1000J_3(2) = 346,2+282,4-129 = 499,6$. Siguiendo éste mismo procedimiento, obtendríamos el resto de las amplitudes finales para cada frecuencia que hemos reunido en la tabla 2.9.

Finalmente, las frecuencias 0 Hz y 1800 Hz no sufrirían superposición alguna, por lo que sus amplitudes no se verán modificadas y valdrán $600J_2(2) = 211,8$ y $1000J_3(2) = 129$, respectivamente.

Frec. (Hz)	0	200	400	600	800	1000	1200	1400	1600	1800
Amp.	211,8	-268,8	31,2	499,6	103,2	-320,4	685,6	859,4	456,2	129

Tabla 2.9: Espectro final obtenido para el ejemplo de modulación FM serie (teniendo en cuenta las supeposiciones).

Para finalizar el apartado, realizaremos la representación correspondiente del espectro resultante:

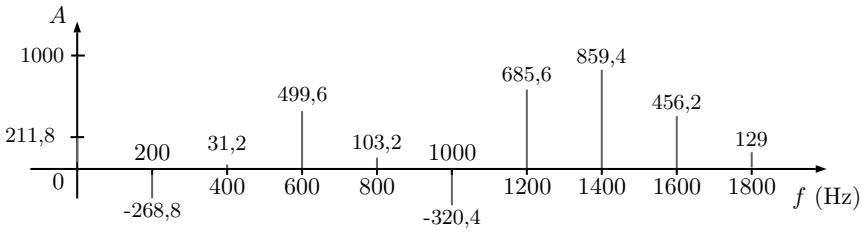


Figura 2.38: Representación del espectro obtenido para el ejemplo de modulación FM serie (teniendo en cuenta las superposiciones).

Para contrastar los cálculos, podemos ver que el espectro es, efectivamente, armónico y existe una frecuencia fundamental de $f_0 = 200$ Hz.

La síntesis FM en serie se suele emplear para conseguir formantes, con f_m como frecuencia fundamental. Si se desea que el espectro obtenido sea armónico, las frecuencias de las N portadoras (f_{c_1}, f_{c_2}, ..., f_{c_N}) deberán coincidir con un armónico de la frecuencia fundamental, que será la frecuencia central de cada formante.

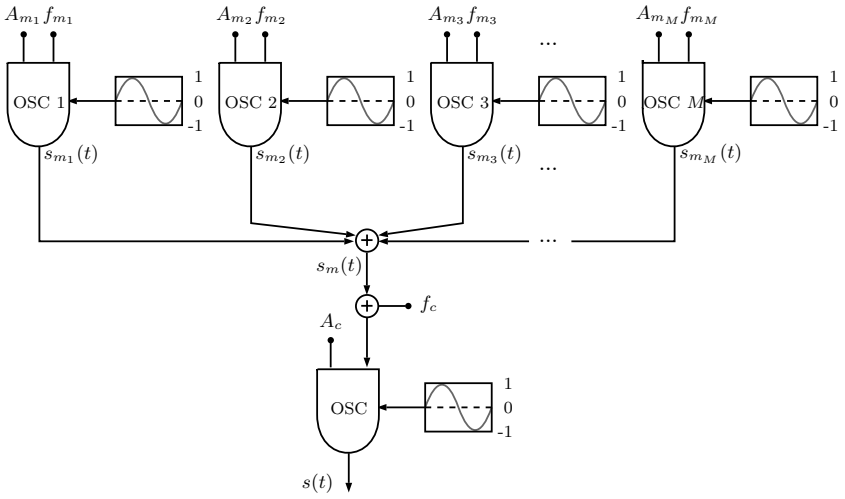
2.3.2.2. Modulación FM en paralelo

La modulación FM en paralelo consiste en el uso de varias señales moduladoras para modular una señal portadora. Por lo tanto, ésto sería equivalente a modular una portadora con una señal compuesta (Fig. 2.39). En éste sentido, cada moduladora sinusoidal generará sus propios parciales, que estarán situados alrededor de cada parcial del resto de moduladoras.

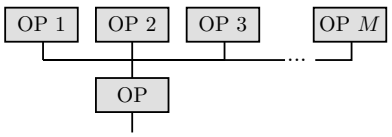
La ecuación que caracteriza la asociación en paralelo para una modulación FM con M señales moduladoras se presenta a continuación:

$$s(t) = A_c \sum_{k_1=-\infty}^{+\infty} ... \sum_{k_M=-\infty}^{+\infty} \left(\prod_{i=1}^{M} J_{k_i}(I_i) \right) \cdot \text{sen}\left(2\pi f_c t + \sum_{i=1}^{M} k_i \cdot 2\pi f_{m_i} t\right) \qquad (2.64)$$

Aunque pueda parecer una ecuación algo confusa, veremos de manera breve que su demostración no es complicada partiendo de lo que ya conocemos.



(a)



(b)

Figura 2.39: Diagrama detallado (a) y simplificado (b) del algoritmo de modulación FM paralelo para M osciladores moduladores.

Por FM simple, sabemos que se cumple que:

$$\text{sen}\left(2\pi f_c t + I\text{sen}(2\pi f_m t)\right) = \sum_{k=-\infty}^{+\infty} J_k(I) \cdot \text{sen}(2\pi f_c t + k \cdot 2\pi f_m t) \quad (2.65)$$

Ahora bien, si en vez de un modulador tenemos por ejemplo dos, la anterior expresión se verá modificada de la siguiente manera:

$$\text{sen}\left(2\pi f_c t + I_1\text{sen}\left(2\pi f_{m_1} t\right) + I_2\text{sen}\left(2\pi f_{m_2} t\right)\right) \quad (2.66)$$

A continuación, si aplicamos a 2.66 la identidad 2.65 con respecto a la primera moduladora, obtendríamos lo siguiente:

$$\sum_{k_1=-\infty}^{+\infty} J_{k_1}(I_1) \cdot \text{sen}\ \left(2\pi f_c t + I_2\text{sen}\left(2\pi f_{m_2} t\right) + k_1 \cdot 2\pi f_{m_1} t\right) \quad (2.67)$$

Finalmente, volvemos a aplicar en la anterior expresión la identidad inicial, ésta vez con respecto a la segunda moduladora:

$$\sum_{k_1=-\infty}^{+\infty} J_{k_1}(I_1) \cdot \left(\sum_{k_2=-\infty}^{+\infty} J_{k_2}(I_2) \cdot \text{sen } (2\pi f_c t + k_1 \cdot 2\pi f_{m_1} t + k_2 \cdot 2\pi f_{m_2} t) \right) \quad (2.68)$$

Dado que el término $J_{k_1}(I_1)$ no depende del segundo sumatorio, lo podemos introducir en él:

$$\sum_{k_1=-\infty}^{+\infty} \sum_{k_2=-\infty}^{+\infty} J_{k_1}(I_1) \cdot J_{k_2}(I_2) \cdot \text{sen } (2\pi f_c t + k_1 \cdot 2\pi f_{m_1} t + k_2 \cdot 2\pi f_{m_2} t) \quad (2.69)$$

Como vemos, si multiplicamos lo anterior por la amplitud de la portadora A_c y en lugar de escoger dos moduladores lo generalizamos para M, conseguiríamos la ecuación de la FM compuesta en paralelo (2.64).

Al exisitir M señales moduladoras, cada una de ellas presentará su frecuencia y amplitud característica (f_{m_i} y A_{m_i}). Por consiguiente, también tendrán su propio índice de modulación I_i, que se calculará como:

$$I_i = \frac{A_{m_i}}{f_{m_i}} \quad (2.70)$$

Con lo cual, el parcial de mayor orden K_i y el número total de parciales P_i que será capaz de generar cada moduladora será:

$$K_i = I_i + 1 \quad (2.71)$$

$$P_i = 2K_i + 1 \quad (2.72)$$

Y el índice k_i del parcial creado por cada moduladora valdrá:

$$k_i = 0, \pm 1, \pm 2, ..., \pm K_i \quad (2.73)$$

Así pues, el número total de parciales P generados en una modulación FM en paralelo dependerá de las M moduladoras, así como de todos los índices de modulación I_i, lo que se traducirá en:

$$P = \prod_{i=1}^{M} P_i = \prod_{i=1}^{M} (2K_i + 1) \quad (2.74)$$

Descontando a P el número de parciales superpuestos, sumando las amplitudes de los mismos con su correspondiente signo.

Las frecuencias de los parciales resultantes serán en éste caso las que se muestran:

$$f_{k_1, k_2, ..., k_M} = f_c + k_1 f_{m_1} + k_2 f_{m_2} + ... + k_M f_{m_M} \quad (2.75)$$

En lo que respecta a las amplitudes de los parciales generados, si observamos la ecuación general de la FM en paralelo, podemos destacar que se corresponderán con la siguiente expresión:

$$A_{k_1,k_2,...,k_M} = A_c \cdot \prod_{i=1}^{M} J_{k_i}(I_i) = A_c \cdot (J_{k_1}(I_1) \cdot J_{k_2}(I_2) \cdot \ ... \ \cdot J_{k_M}(I_M)) \quad (2.76)$$

Pongamos la teoría vista en práctica mediante el siguiente ejemplo.

Ejemplo: Determine el espectro correspondiente a una modulación FM compuesta en paralelo con dos señales moduladoras y una portadora. Datos de las señales: $A_c = 1000$, $f_c = 1000$ Hz, $A_{m_1} = 500$ Hz, $f_{m_1} = 250$ Hz, $A_{m_2} = 600$ Hz y $f_{m_2} = 200$ Hz.

Empezamos calculando los índices de modulación de ambas moduladoras:

$$I_1 = \frac{A_{m_1}}{f_{m_1}} = \frac{500\,\text{Hz}}{250\,\text{Hz}} = 2 \quad (2.77)$$

$$I_2 = \frac{A_{m_2}}{f_{m_2}} = \frac{600\ \text{Hz}}{200\ \text{Hz}} = 3 \quad (2.78)$$

Seguidamente, obtenemos los parciales de mayor orden y el número total de parciales que generará cada moduladora:

$$K_1 = I_1 + 1 = 2 + 1 = 3 \quad (2.79)$$

$$K_2 = I_2 + 1 = 3 + 1 = 4 \quad (2.80)$$

$$P_1 = 2K_1 + 1 = 6 + 1 = 7 \text{ parciales} \quad (2.81)$$

$$P_2 = 2K_2 + 1 = 8 + 1 = 9 \text{ parciales} \quad (2.82)$$

Por lo tanto, el número total de parciales debido a la modulación FM en paralelo sin contar superposiciones será:

$$P = P_1 \cdot P_2 = 7 \cdot 9 = 63 \text{ parciales} \quad (2.83)$$

Llegados a éste punto, procedemos al cálculo de las frecuencias de los parciales a partir de la expresión que sigue:

$$f_{k_1,\,k_2} = f_c + k_1 f_{m_1} + k_2 f_{m_2} \quad (2.84)$$

Donde los índices k_1 y k_2 tomarán los siguiente valores:

$$k_1 = 0,\ \pm1,\ \pm2,\ \pm3 \quad (2.85)$$

$$k_2 = 0,\ \pm1,\ \pm2,\ \pm3,\ \pm4 \quad (2.86)$$

Las amplitudes de los parciales generados se podrán determinar según se muestra:

$$A_{k_1,\,k_2} = A_c \cdot J_{k_1}(I_1) \cdot J_{k_2}(I_2) \tag{2.87}$$

Por ejemplo, si deseamos calcular la frecuencia del parcial cuyos índices son $k_1 = -1$ y $k_2 = 3$, lo haríamos del siguiente modo: $f_{-1,3} = 1000 - 1 \cdot 250 + 3 \cdot 200 = 1350$ Hz. Si quisiéramos determinar la frecuencia del parcial de índices $k_1 = 2$ y $k_2 = -4$ sería $f_{2,-4} = 1000 + 2 \cdot 250 - 4 \cdot 200 = 700$ Hz. Siguiendo el procedimiento para el resto de parciales, conseguiríamos averiguar todas las frecuencias.

En el caso de las amplitudes, para calcular las correspondientes a los anteriores parciales, tendríamos que $A_{-1,3} = 1000 \cdot J_{-1}(2) \cdot J_3(3) = 1000 \cdot (-0,577) \cdot 0,309 = -178,3$ y $A_{2,-4} = 1000 \cdot J_2(2) \cdot J_{-4}(3) = 1000 \cdot 0,353 \cdot 0,132 = 46,6$.

Creamos dos tablas para reunir todas las frecuencias y amplitudes de cada parcial en función de los índices k_1 y k_2:

k_1 \ k_2	-4	-3	-2	-1	0	1	2	3	4
-3	-550	-350	-150	50	250	**450**	**650**	**850**	**1050**
-2	-300	-100	100	300	500	**700**	**900**	**1100**	**1300**
-1	-50	150	350	550	750	**950**	**1150**	**1350**	**1550**
0	200	400	600	800	1000	1200	1400	1600	1800
1	**450**	**650**	**850**	**1050**	1250	1450	1650	1850	2050
2	**700**	**900**	**1100**	**1300**	1500	1700	1900	2100	2300
3	**950**	**1150**	**1350**	**1550**	1750	1950	2150	2350	2550

Tabla 2.10: Frecuencias de los parciales generados para el ejemplo de modulación FM en paralelo (señalando las superposiciones).

k_1 \ k_2	-4	-3	-2	-1	0	1	2	3	4
-3	-17,0	39,9	-62,7	43,7	33,5	-43,7	-62,7	-39,9	-17,0
-2	46,6	-109,1	171,6	-119,7	-91,8	119,7	171,6	109,1	46,6
-1	-76,2	178,3	-280,4	195,6	150,0	-195,6	-280,4	-178,3	-76,2
0	29,6	-69,2	108,9	-75,9	-58,2	75,9	108,9	69,2	29,6
1	76,2	-178,3	280,4	-195,6	-150,0	195,6	280,4	178,3	76,2
2	46,6	-109,1	171,6	-119,7	-91,8	119,7	171,6	109,1	46,6
3	17,0	-39,9	62,7	-43,7	-33,5	43,7	62,7	39,9	17,0

Tabla 2.11: Amplitudes de los parciales generados para el ejemplo de modulación FM en paralelo (multiplicadas por A_c y sin contar superposiciones).

Antes de poder representar el espectro, debemos resolver las superposiciones producidas entre los parciales. Del mismo modo que con el algoritmo

FM serie, se producen dos tipos de superposiciones (tabla 2.10): por diferencia de fase de 180º (subrayadas) y por coincidir en frecuencia (en negrita).

En el caso de los parciales que difieren en la fase, como ocurre por ejemplo para las frecuencias de -550 Hz y 550 Hz, habría que pasar la frecuencia -550 Hz a positiva, cambiando el signo de la amplitud del parcial y sumándola a la amplitud de 550 Hz. Con ello, tendríamos la amplitud final para la frecuencia de 550 Hz: $A_{(550\,\mathrm{Hz})} = (+)17,0 + 195,6 = 212,6$. También tendríamos otra superposición de éste tipo para las frecuencias -350 Hz y 350 Hz, que resolviéndola quedaría: $A_{(350\,\mathrm{Hz})} = (-)39,9 - 280,4 = -320,3$.

Las superposiciones en las que coinciden directamente las frecuencias de los parciales se resolverían sumándolas. Así pues, para la frecuencia de 1550 Hz, existirían dos parciales que coinciden: el $f_{-1,4}$ y el $f_{3,-1}$. Por tanto, la amplitud final que tendrá en el espectro la frecuencia de 1550 Hz será: $A_{(1550\,\mathrm{Hz})} = -76,2 - 43,7 = -119,9$. Otra de las frecuencias para la que existe superposición podría ser 950 Hz, coincidiendo los parciales $f_{3,-4}$ y $f_{-1,1}$: $A_{(950\,\mathrm{Hz})} = -195,6 + 17,0 = -178,6$.

Las frecuencias que no sufren superposición, como ocurre con 1000 Hz, tendrían como valor de amplitud final la que se muestra en la tabla 2.11 (para 1000 Hz sería $-58,2$).

Agrupamos las amplitudes finales de los parciales (con las superposiciones resueltas) en función de las frecuencias para poder representar finalmente el espectro (Fig. 2.40):

Frec. (Hz)	50	100	150	200	250	300	350	400	450
Amp.	119,9	280,7	241,0	29,6	33,5	-166,3	-320,3	-69,2	32,5
Frec. (Hz)	500	550	600	650	700	750	800	850	900
Amp.	-91,8	212,6	108,9	-241,0	166,3	150,0	-75,9	240,5	62,5
Frec. (Hz)	950	1000	1050	1100	1150	1200	1250	1300	1350
Amp.	-178,6	-58,2	-212,6	280,7	-320,3	75,9	-150,0	-73,1	-115,6
Frec. (Hz)	1400	1450	1500	1550	1600	1650	1700	1750	1800
Amp.	108,9	195,6	-91,8	-119,9	69,2	280,4	119,7	-33,5	29,6
Frec. (Hz)	1850	1900	1950	2050	2100	2150	2300	2350	2550
Amp.	178,3	171,6	43,7	76,2	109,1	62,7	46,6	39,9	17,0

Tabla 2.12: Espectro final obtenido para el ejemplo de modulación FM paralelo (teniendo en cuenta las superposiciones).

Éste ejemplo nos ha servido para ver que el algoritmo de modulación FM paralelo permite crear espectros muy cargados en parciales utilizando pocos osciladores.

El posible inconveniente de éste algoritmo sería que con más de dos señales moduladoras se hace muy difícil de controlar. Además de ello, los

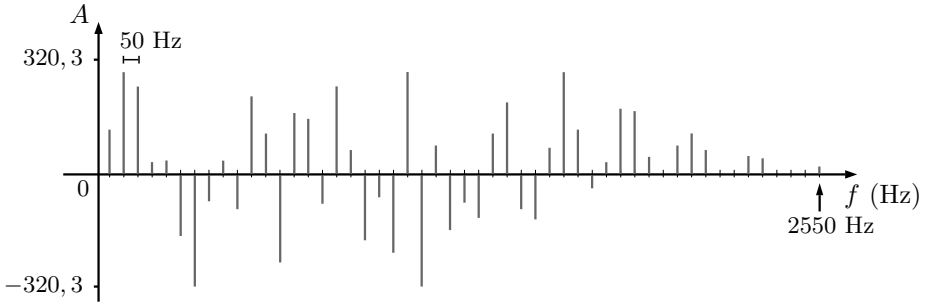


Figura 2.40: Representación del espectro final para el ejemplo de modulación FM paralelo (teniendo en cuenta las superposiciones).

índices de modulación deben presentar valores muy bajos, ya que en caso contrario, la señal de salida será muy ruidosa debido a la gran cantidad de parciales creados.

La modulación FM paralelo puede resultar interesante en momentos específicos del sonido que queremos diseñar, como por ejemplo en el tiempo de ataque de un sonido de tipo percusivo.

2.3.2.3. Modulación FM en cascada

Se produce modulación FM en cascada cuando una señal FM se emplea como moduladora de una señal portadora. En éste sentido, cada oscilador modularía al siguiente al que está conectado (véase Fig. 2.41). La ecuación que nos proporciona información sobre el espectro que se va a generar cuando existen M moduladoras en una configuración FM en cascada sería la siguiente:

$$s(t) = A_c \sum_{k_1=-\infty}^{+\infty} ... \sum_{k_M=-\infty}^{+\infty} J_{k_1}(I_1) \cdot \left(\prod_{i=2}^{M} J_{k_i}(k_{i-1} \cdot I_i) \right) \cdot \text{sen}\,(2\pi f_c t + \sum_{i=1}^{M} k_i \cdot 2\pi f_{m_i} t) \tag{2.88}$$

La demostración de dicha ecuación se llevaría a cabo del mismo modo que en el caso de modulación FM en paralelo. Si estudiamos la ecuación, podremos apreciar que el número total de parciales generados, así como sus frecuencias asociadas serán las mismas que en el caso del algoritmo paralelo, con lo que únicamente cambia el cálculo de las amplitudes:

$$A_{k_1, k_2, ..., k_M} = A_c \cdot J_{k_1}(I_1) \cdot \prod_{i=2}^{M} J_{k_i}(k_{i-1} I_i) \tag{2.89}$$

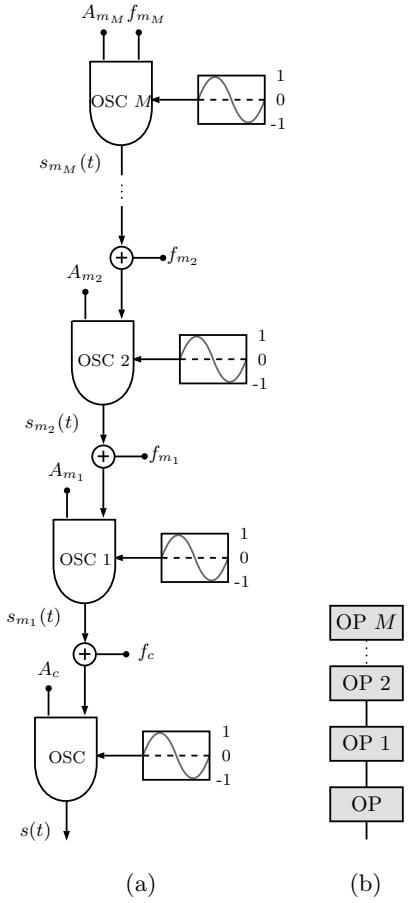


Figura 2.41: Diagrama detallado (a) y simplificado (b) del algoritmo de modulación FM cascada para M osciladores moduladores.

Nótese que cada banda de un modulador (k_{i-1}) se emplea para escalar el índice de modulación del siguiente (I_i).

Las principales diferencias del algoritmo FM cascada con respecto al FM paralelo, utilizando los mismos valores de índice de modulación y relaciones de frecuencia son:

- Las mismas bandas de frecuencia tienen en promedio menor amplitud. Fijémonos que como en éste caso se produce un escalado del índice de modulación a partir del 2º modulador ($i = 2$), el argumento de la función de *Bessel* correspondiente J_{k_i} será mayor para cascada ($k_{i-1}I_i$) que para paralelo (I_i). Con lo cual, dado que las funciones de *Bessel* disminuyen a medida que aumenta el valor del argumento, la amplitud será menor.

- A partir del 2º modulador ($i = 2$), los parciales cuyos índices cumplen que $k_{i-1} = 0$ y $k_i \neq 0$ tienen amplitud nula. En efecto, si consideramos un ejemplo para dos moduladores y un portador, las amplitudes de los parciales generados serán: $A_{k_1,\,k_2} = A_c \cdot J_{k_1}(I_1) \cdot J_{k_2}(k_1 I_2)$. Cuando el índice k_1 valga cero, las amplitudes de los parciales dependerán del índice k_2 y serán: $A_{0,\,k_2} = A_c \cdot J_0(I_1) \cdot J_{k_2}(0)$. En ésta última expresión, vemos que el término $J_{k_2}(0)$ valdrá 1 cuando $k_2 = 0$, ya que $J_0(0) = 1$. En cambio, cuando $k_2 \neq 0$, $J_{k_2}(0)$ valdrá 0. Por consiguiente, la amplitud del parcial creado será nula cuando $k_1 = 0$ y $k_2 \neq 0$.

Veamos seguidamente un ejemplo con los mismos datos que en el caso del algoritmo de FM en paralelo, a fin de apreciar las diferencias entre ambos algoritmos.

Ejemplo: Determine el espectro correspondiente a una modulación FM compuesta en cascada con dos señales moduladoras y una portadora. Datos de las señales: $A_c = 1000$, $f_c = 1000$ Hz, $A_{m_1} = 500$ Hz, $f_{m_1} = 250$ Hz, $A_{m_2} = 600$ Hz y $f_{m_2} = 200$ Hz.

En éste caso, los índices de modulación serán los mismos que en caso de síntesis FM en paralelo:

$$I_1 = \frac{A_{m_1}}{f_{m_1}} = \frac{500 \text{ Hz}}{250 \text{ Hz}} = 2 \tag{2.90}$$

$$I_2 = \frac{A_{m_2}}{f_{m_2}} = \frac{600 \text{ Hz}}{200 \text{ Hz}} = 3 \tag{2.91}$$

Del mismo modo, el parcial de mayor orden y número total de parciales por moduladora será también idéntico que en el caso paralelo:

$$K_1 = I_1 + 1 = 2 + 1 = 3 \tag{2.92}$$

$$K_2 = I_2 + 1 = 3 + 1 = 4 \tag{2.93}$$

$$P_1 = 2K_1 + 1 = 6 + 1 = 7 \text{ parciales} \tag{2.94}$$

$$P_2 = 2K_2 + 1 = 8 + 1 = 9 \text{ parciales} \tag{2.95}$$

El número total de parciales generados en el espectro será:

$$P = P_1 \cdot P_2 = 7 \cdot 9 = 63 \text{ parciales} \tag{2.96}$$

Pero según lo comentado anteriormente, las amplitudes de algunos parciales serán nulas cuando $k_1 = 0$ y $k_2 \neq 0$, con lo que no podremos contar los 8 parciales de frecuencias $f_{0,\,k_2}$ que cumplen ésta condición, por lo que tendremos $P = 63 - 8 = 55$ parciales totales.

Las frecuencias también se calcularán como en el apartado anterior,

y como ya mencionamos, la principal diferencia con respecto al caso paralelo será el cálculo de las amplitudes:

$$f_{k_1,\,k_2} = f_c + k_1 f_{m_1} + k_2 f_{m_2} \tag{2.97}$$

$$A_{k_1,k_2} = A_c \cdot J_{k_1}(I_1) \cdot J_{k_2}(k_1 I_2) \tag{2.98}$$

Donde los índices k_1 y k_2 valdrán:

$$k_1 = 0,\ \pm 1,\ \pm 2,\ \pm 3 \tag{2.99}$$

$$k_2 = 0,\ \pm 1,\ \pm 2,\ \pm 3,\ \pm 4 \tag{2.100}$$

La amplitud, por ejemplo, del parcial de índices $k_1 = -3$ y $k_2 = 1$ sería: $A_{-3,1} = 1000 \cdot J_{-3}(2) \cdot J_1(-3 \cdot 3) = 1000 \cdot (-J_3(2)) \cdot (J_1(-9)) = 1000 \cdot (-0,129) \cdot (-0,245) = 31,6$; y su frecuencia tendrá un valor de: $f_{-3,\,1} = 1000 - 3 \cdot 250 + 1 \cdot 200 = 450$ Hz. Si comparamos ésta amplitud con respecto a la amplitud análoga en paralelo, vemos que en cascada se consigue una magnitud inferior ($|31,6| = 31,6$) con respecto al caso paralelo ($|-43,7| = 43,7$).

Es importante darse cuenta que, hasta éste momento, no nos habíamos encontrado con funciones de *Bessel* que presentasen índice de modulación negativo, como nos acaba de ocurrir cuando hemos tenido que calcular $J_1(-9)$. Cuando una función de *Bessel* de 1ª especie presenta un argumento negativo, debemos actuar de la misma forma que cuando el índice k era negativo: $J_k(-I) = (-1)^k \cdot J_k(I)$ (con $I > 0$).

En lo que respecta a las superposiciones, se tratarían de igual forma que en el resto de casos ya estudiados.

Si continuamos calculando las amplitudes y frecuencias del resto de parciales, obtendríamos las siguientes tablas:

k_1 \ k_2	-4	-3	-2	-1	0	1	2	3	4
-3	-550	-350	-150	50	250	**450**	**650**	**850**	**1050**
-2	-300	-100	100	300	500	**700**	**900**	**1100**	**1300**
-1	-50	150	350	550	750	**950**	**1150**	**1350**	**1550**
0	–	–	–	–	1000	–	–	–	–
1	**450**	**650**	**850**	**1050**	1250	1450	1650	1850	2050
2	**700**	**900**	**1100**	**1300**	1500	1700	1900	2100	2300
3	**950**	**1150**	**1350**	**1550**	1750	1950	2150	2350	2550

Tabla 2.13: Frecuencias de los parciales generados para el ejemplo de modulación FM en cascada (señalando las superposiciones y representando con un guión aquellas frecuencias de amplitud nula).

k_1 \ k_2	-4	-3	-2	-1	0	1	2	3	4
-3	34,2	23,3	-18,7	-31,6	11,6	31,6	-18,7	-23,3	34,2
-2	126,2	40,5	-85,7	-97,6	53,2	97,6	-85,7	-40,5	126,2
-1	-76,1	-178,2	-280,3	-195,5	150,0	195,5	-280,3	178,2	-76,1
0	0,0	0,0	0,0	0,0	223,9	0,0	0,0	0,0	0,0
1	76,1	-178,2	280,3	-195,5	-150,0	195,5	280,3	178,2	76,1
2	126,2	-40,5	-85,7	97,6	53,2	-97,6	-85,7	40,5	126,2
3	-34,2	23,3	18,7	-31,6	-11,6	31,6	18,7	-23,3	-34,2

Tabla 2.14: Amplitudes de los parciales generados para el ejemplo de modulación FM en cascada (multiplicadas por A_c y sin contar superposiciones).

Finalmente, resolvemos las superposiciones para poder obtener la representación del espectro final (Fig. 2.42):

Frec. (Hz)	50	100	150	200	250	300	350	400	450
Amp.	44,5	-126,2	-159,5	0,0	11,6	-223,8	-303,6	0,0	107,7
Frec. (Hz)	500	550	600	650	700	750	800	850	900
Amp.	53,2	-229,7	0,0	-196,9	223,8	150,0	0,0	257,0	-126,2
Frec. (Hz)	950	1000	1050	1100	1150	1200	1250	1300	1350
Amp.	161,3	223,9	-161,3	-126,2	-257,0	0,0	-150,0	223,8	196,9
Frec. (Hz)	1400	1450	1500	1550	1600	1650	1700	1750	1800
Amp.	0,0	195,5	53,2	-107,7	0,0	280,3	-97,6	-11,6	0,0
Frec. (Hz)	1850	1900	1950	2050	2100	2150	2300	2350	2550
Amp.	178,2	-85,7	31,6	76,1	40,5	18,7	126,2	-23,3	-34,2

Tabla 2.15: Espectro final obtenido para el ejemplo de modulación FM cascada (teniendo en cuenta las supeposiciones).

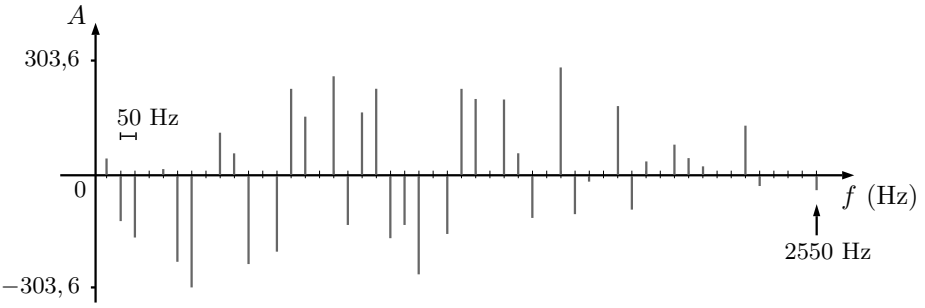


Figura 2.42: Representación del espectro final para el ejemplo de modulación FM cascada (teniendo en cuenta las superposiciones).

2.3.2.4. Modulación FM con realimentación

En la modulación FM con realimentación, se conecta la salida del oscilador a su propia entrada. Dicha salida se multiplica por el índice de modulación antes de ser sumada a la frecuencia fundamental del oscilador. Por

tanto, el oscilador actúa de portador y modulador simultáneamente (véase Fig. 2.43).

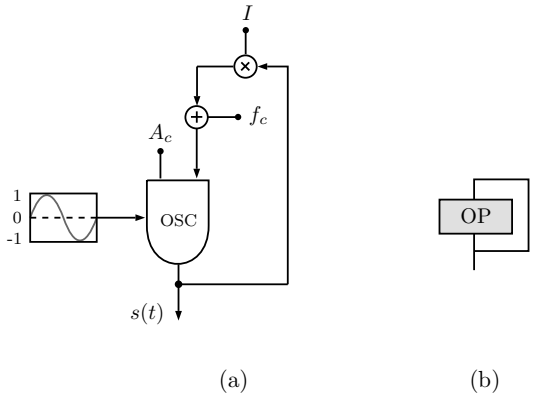


Figura 2.43: Diagrama detallado (a) y simplificado (b) del algoritmo de modulación FM con realimentación.

Se puede demostrar que la ecuación que define el espectro resultante para éste algoritmo es:

$$s(t) = A_c \sum_{k=1}^{+\infty} \frac{2}{kI} J_k(kI) \cdot \text{sen}\,(k \cdot 2\pi f_c t) \tag{2.101}$$

A diferencia de los anteriores algoritmos estudiados, la ecuación para la FM con realimentación presenta dos novedades:

- Solo aparecen en el espectro los múltiplos enteros positivos de la frecuencia de portadora (f_c), ya que $k = 1, 2, ..., +\infty$.
- Encontramos un nuevo factor: $2/kI$.

Por otra parte, como ocurría con FM en cascada, el argumento de la función de *Bessel* depende del índice k ($J_k(kI)$).

Cada uno de los cambios mencionados provocará una consecuencia en el espectro:

- El espectro obtenido siempre será armónico debido a que solo existen múltiplos enteros positivos de f_c. Por tanto, la frecuencia fundamental de éste espectro será $f_0 = f_c$.
- El factor $2/kI$ provoca que tengamos un espectro mucho más ancho y plano con respecto al resto de algoritmos FM. Como vemos, la amplitud de los parciales bajará progresivamente debido a que es inversamente proporcional al índice del parcial k ($A_p \propto 1/k$), con lo que a la salida del oscilador tendremos una señal tipo diente de sierra.

El índice de modulación para FM con realimentación (o *factor de realimentación*) también determinará el número de parciales generados. No obstante, dicho parámetro será mucho más sensible a variaciones que en los algoritmos anteriores (donde habían osciladores no realimentados). En éste sentido, la bibliografía sugiere para I el uso de valores comprendidos entre 0 y $1,5$. Un valor superior a $1,5$ (por ejemplo $I = 2$) haría que se generase ruido blanco debido a la gran cantidad de parciales generados.

Como dato característico, para $I = 1$, los 32 primeros armónicos presentarían una amplitud superior al $1\,\%$ (-40 dB) de la amplitud de la fundamental. Por otra parte, para $I = 0,5$, el sexto armónico ($k = 6$) tendría una amplitud de -42 dB. Éste hecho nos da una idea de la extrema sensibilidad que tiene el índice de modulación.

Si continuamos observando la ecuación 2.101, podemos observar que el índice de modulación I se encuentra en el denominador (factor $2/kI$), luego podría ser un caso problemático si I se aproxima mucho a cero. No obstante, se puede demostrar que:

$$\lim_{I \to 0} \frac{2}{kI} J_k(kI) = \begin{cases} 1 \text{ si } k = 1 \\ 0 \text{ si } k > 1 \end{cases} \tag{2.102}$$

Es decir, cuando $I = 0$ tendremos a la salida del oscilador una señal sinusoidal, como en el resto de algoritmos FM.

Como conclusión, las ventajas que aporta la síntesis FM realimentada con respecto a la FM simple es que si deseamos generar un sonido armónico, tendremos la máxima eficiencia ya que con un único oscilador seremos capaces de sintetizar un número arbitrario de parciales. Además, el decaimiento de la amplitud de los parciales será lineal, por lo que tendremos un timbre generado que será menos natural, pudiendo ser interesante si queremos crear nuevos sonidos.

Para finalizar el apartado, veamos un ejemplo de síntesis FM con realimentación.

Ejemplo: Obtenga el espectro de los 10 primeros parciales para una modulación FM con realimentación para un único oscilador con $A_c = 100$, $f_c = 500$ Hz e $I = 1$.

Las frecuencias f_k para éste caso se calcularían de la forma:

$$f_k = k \cdot f_c \tag{2.103}$$

Y las amplitudes se determinarán mediante la expresión:

$$A_k = A_c \cdot \frac{2}{kI} \cdot J_k(kI) \tag{2.104}$$

Donde k tomará los valores:

$$k = 1, 2, \ldots, +\infty \tag{2.105}$$

A continuación, agrupamos en una tabla las frecuencias y amplitudes para los 10 primeros parciales:

Parcial (k)	1	2	3	4	5	6	7	8	9	10
Frecuencia (Hz)	500	1000	1500	2000	2500	3000	3500	4000	4500	5000
Amplitud ($\times A_c$)	88,0	35,3	20,6	14,1	10,4	8,2	6,7	5,6	4,8	4,1

Tabla 2.16: Espectro final obtenido para el ejemplo de modulación FM con realimentación.

Como vemos, no se producen superposiciones entre parciales. Por último, llevamos a cabo la representación del espectro:

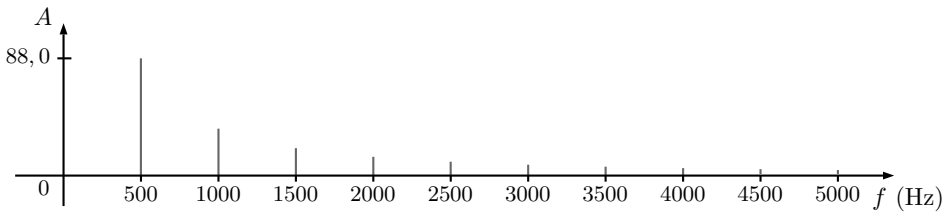


Figura 2.44: Representación del espectro final para el ejemplo de modulación FM con realimentación.

Podemos destacar que la amplitud de los parciales disminuye de manera inversamente proporcional al índice k, por lo que obtendríamos una señal tipo diente de sierra cuya frecuencia fundamental será de $f_0 = 500$ Hz.

2.3.3. Ejemplos sonoros de síntesis FM

Para finalizar éste segundo capítulo, hemos creído oportuno poner a disposición del lector varios ejemplos sonoros que ilustren de un modo intuitivo tanto la síntesis FM simple como la compuesta.

Como primer ejemplo, realizamos síntesis FM simple, partiendo de una amplitud de portadora de $A_c = 0$ dB y modificando la amplitud de la moduladora desde $A_m = -\infty$ hasta $A_m = 0$ dB, pudiendo observar cómo se lleva a cabo la modulación progresivamente. Las frecuencias escogidas han sido $f_c = 800$ Hz y $f_m = 200$ Hz:

10.-Síntesis FM simple.mp3

En el segundo ejemplo, mostramos un caso particular de algoritmo de FM

en serie, con una señal moduladora ($A_m = 0$ dB y $f_m = 400$ Hz) y 3 señales portadoras cuyos datos son: $A_{c_1} = A_{c_2} = A_{c_3} = 0$ dB, $f_{c_1} = 800$ Hz, $f_{c_2} = 600$ Hz y $f_{c_3} = 200$ Hz:

11.-Síntesis FM serie.mp3

A continuación, empleamos el algoritmo de síntesis FM en paralelo, utilizando 3 señales moduladoras y una portadora. Las moduladoras tendrán $A_{m_1} = A_{m_2} = A_{m_3} = 0$ dB, $f_{m_1} = 200$ Hz, $f_{m_2} = 400$ Hz y $f_{m_3} = 600$ Hz. Para la portadora, elegimos $A_c = 0$ dB y $f_c = 800$ Hz:

12.-Síntesis FM paralelo.mp3

Seguimos con otro ejemplo, haciendo uso del algoritmo FM en cascada, donde también habrán 3 moduladoras con la misma amplitud (0 dB) y frecuencias: $f_{m_1} = 600$ Hz, $f_{m_2} = 400$ Hz y $f_{m_3} = 200$ Hz. La portadora tendrá la misma amplitud y frecuencia que en el ejemplo anterior.

13.-Síntesis FM cascada.mp3

Como último ejemplo, tratamos la síntesis FM con realimentación para un oscilador con $A_c = 0$ dB y frecuencia fundamental $f_c = 500$ Hz. Empezamos con un índice de modulación $I = 0$ y vamos subiendo dicho parámetro hasta que se aprecia una señal tipo diente de sierra, alcanzando finalmente ruido blanco:

14.-Síntesis FM con realimentación.mp3

Conocidos ya los fundamentos básicos sobre la síntesis FM, pasaremos a estudiar el lenguaje de programación que utilizaremos para llevar a cabo la implementación de nuestro sintetizador FM.

Capítulo 3

Csound

3.1. Introducción a Csound

Csound es un generador y procesador de sonido digital que se manipula como un lenguaje de programación y permite emular cualquier sintetizador o módulo de efectos comercial, siempre y cuando las características *hardware* del ordenador lo permitan.

La primera versión de *Csound* fue creada en 1985 por *Barry Vercoe* en el MIT (*Massachussets Institute of Technology*), y a día de hoy, continúan lanzando nuevas actualizaciones, alcanzando la versión 5.19 (para más información, visite `www.csounds.com`).

Entre las principales ventajas de ésta herramienta, podemos destacar que se trata de un lenguaje de código abierto, de libre distribución y multiplataforma.

Para poder generar o sintetizar sonido, *Csound* se basa en la compilación de dos ficheros: la *orquesta* y la *partitura*. El fichero de orquesta (de extensión `.orc`) reunirá las definiciones de los distintos instrumentos (sintetizadores, módulos de efectos,...etc), mientras que el fichero de partitura (extensión `.sco`) agrupa las instrucciones que controlarán los instrumentos de la orquesta (notas, tempo,...etc). El proceso básico que sigue *Csound* para la generación del sonido puede observarse en la Fig. 3.1.

Analizaremos seguidamente la estructura y sintaxis básica de los ficheros mencionados para comprender cómo implementar sintetizadores básicos, ahondando también en otras características del lenguaje.

También debemos tener en cuenta que para poder utilizar *Csound* como tal, haremos uso de un *front-end* o interfaz que no solo nos servirá de

editor gráfico para poder programar de un modo cómodo, sino que podremos utilizar los sintetizadores en tiempo real. Aunque existen diversos *front-end* para *Csound*, nosotros utilizaremos uno de los más destacados: *QuteCsound*. Éste programa permitirá el uso de *widgets* o elementos gráficos mediante los que podremos interaccionar con el sintetizador.

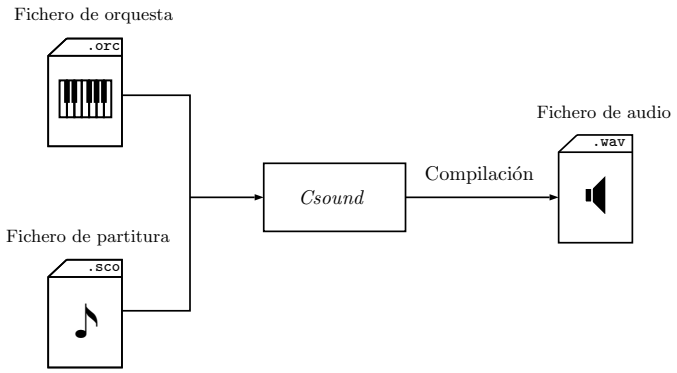


Figura 3.1: Proceso básico de generación de sonido mediante *Csound*.

3.2. Fichero de orquesta (.orc)

Acabamos de comentar que en el fichero de orquesta podremos implementar todos los instrumentos que deseemos. Éste fichero (al igual que el de partitura) está compuesto de líneas de texto correspondientes a las instrucciones que seguirá el sintetizador. La estructura de la orquesta presenta dos bloques:

- *Cabecera*: Está formada por las variables globales reservadas, así como el resto de posibles inicializaciones de otras variables.
- *Cuerpo*: Contiene las definiciones de los instrumentos y otras posibles sentencias.

La figura 3.2 muestra un fichero genérico de orquesta, señalando los bloques mencionados.

3.2.1. Variables globales reservadas

Las *variables globales reservadas* son variables situadas en la cabecera de la orquesta y sirven para establecer valores a distintos parámetros, tales como la frecuencia de muestreo o el número de canales de salida (entre otros), afectando por tanto a las señales generadas por los instrumentos. Éstas variables presentan valores por defecto, por lo que si las eludimos en el código, *Csound* asignará automáticamente el valor por defecto. Las principales variables globales reservadas en *Csound* son las siguientes:

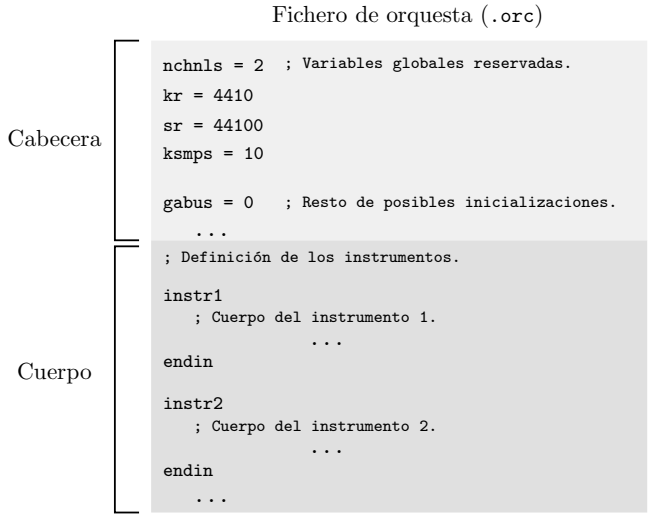


Figura 3.2: Fichero genérico de orquesta en *Csound*.

- `nchnls`: Nos permite establecer el *número de canales de salida* que tendrá el archivo de audio generado. Así pues, dependiendo del valor que tome `nchnls`, el sonido final podrá ser monofónico (`nchnls = 1`), estereofónico (`nchnls = 2`) o cuadrafónico (`nchnls = 4`). Por defecto, *Csound* establece como valor predeterminado `nchnls = 1`.
- `sr`: Ésta variable nos permite modificar la *frecuencia de muestreo* (en Hz). Por defecto, la variable `sr` toma el valor `44100` Hz.
- `kr`: Se trata de la *frecuencia de control* (también medida en Hz) y nos indica cuán rápido se actualizan las variables de control (más adelante estudiaremos éste tipo de variables). Por definición, se establece `kr = 4410` Hz.
- `ksmps`: Es el *número de muestras* que se calculan antes de cambiar los valores de control. También se define como `ksmps = sr/kr`. Cuanto mayor sea el valor de ésta variable global, mayor calidad tendrá el sonido final. Por defecto, `ksmps = 10` muestras.

También existen otras variables globales reservadas como por ejemplo `0dbfs`, para fijar el valor de amplitud PCM[1] que se considerará 0 dB. Por defecto, tenemos `0dbfs = 32767`, que se corresponde con la amplitud máxima de una señal bipolar de 16 bits.

[1] *Pulse-Code Modulation*: Las muestras se codifican por su amplitud mediante números enteros.

3.2.2. Definición de instrumentos

Pasando al cuerpo de la orquesta, podemos observar que se lleva a cabo la definición de los instrumentos. Un *instrumento* puede considerarse como un bloque capaz de generar o procesar señales, dependiendo del mecanismo que implementemos en ellos, a través de la definición e interconexión de distintos módulos (funciones). Un sintetizador puede estar compuesto por uno o más instrumentos. La sintaxis que siguen es la siguiente:

```
instr NÚMERO DE IDENTIFICACIÓN
    ; Sentencias de especificación del instrumento.
    ; Definiciones.
    ; Opcodes (operation codes).
    ...
endin
```

Figura 3.3: Sintaxis para la definición de instrumentos en *Csound*.

Para comenzar la definición del instrumento, escribimos la palabra reservada `instr` seguida del número de identificación, que caracteriza de manera unívoca al instrumento. Seguidamente, se escriben todas las instrucciones que deseemos que ejecute el instrumento y finalizamos su definición con `endin`. Entre los distintos tipos de instrucciones, encontramos los *opcodes* (*operation codes*).

Un *opcode* es una función que contiene un identificador (nombre) y presenta uno o varios parámetros de entrada y de salida. A efectos de síntesis, puede considerarse un *módulo* que genera o modifica señales. La sintaxis de un *opcode* es la que se muestra a continuación:

```
(salida1, salida2,..., salidaN)   OPCODE   (entrada1, entrada2,..., entradaN)
```

Figura 3.4: Sintaxis para la definición de un *opcode* en *Csound*.

Como puede observarse, los parámetros de salida se colocan a la derecha del identificador del opcode (`OPCODE`), mientras que los parámetros de entrada se colocan a la izquierda. Tanto los parámetros de entrada como los de salida deben ir separados por comas siempre que exista más de uno. En ocasiones, algunos parámetros pueden ser opcionales. *Csound* dispone de una extensa librería de *opcodes* enfocados a muchas técnicas de síntesis de sonido (véase [8]). Los comentarios en *Csound* se escriben con el símbolo `;`, y mediante la barra `\` al final de una línea, podemos continuarla en el siguiente renglón. A continuación, vamos a ver un ejemplo de definición de instrumento en *Csound*.

Ejemplo: Implementar un oscilador sinusoidal en *Csound* que tenga amplitud $A = 20000$ y frecuencia $f = 500$ Hz.

Una posible solución para éste ejemplo sería la que se muestra a continuación:

```
; Definición del instrumento (oscilador sinusoidal).
instr 1
      asalida oscil 20000, 500, 101 ; Generación de la señal "asalida".
      out asalida                   ; Salida de la señal "asalida".
endin
```

Figura 3.5: Ejemplo de implementación de un oscilador sinusoidal en *Csound*.

Comenzamos definiendo el instrumento con número de identificación `1`. A continuación, generamos la señal sinusoidal haciendo uso del *opcode* `oscil`. Éste *opcode* se encarga de generar una señal periódica a partir de la repetición de una tabla de onda según la frecuencia y amplitud introducida[2].

De éste modo, creamos la señal sinusoidal de amplitud `20000`[3], frecuencia fundamental `500` Hz y una tabla que contiene un ciclo de onda sinusoidal (tabla `101`, creada en la partitura como veremos más adelante).

El resultado de la onda generada se almacena en la variable `asalida`. Finalmente, se manda la señal monofónica `asalida` a la salida de la tarjeta de sonido mediante el opcode `out` (en caso de ser salida estéreo, necesitaríamos emplear el *opcode* `outs`). Además, el sonido generado puede ser grabado en un fichero de audio (por ejemplo un `.wav`).

La notación gráfica que vimos en el capítulo 2 también se emplea en *Csound*, solo que ahora nombraremos los dispositivos (osciladores, amplificadores, filtros...etc) con el identificador del *opcode* al que se haga referencia. El diagrama referente al ejemplo se adjunta a continuación:

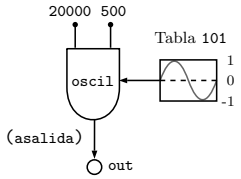


Figura 3.6: Diagrama básico de un oscilador sinusoidal en *Csound*.

[2]También existen otros *opcodes* que generan señales periódicas como: `oscili`, `oscil3`, `poscil`, etc.

[3]Una amplitud de 20000 PCM para una señal de 16 bit equivaldría a una amplitud normalizada de 20000/32767=0,61.

3.2.3. Tipos de variables

Como todo lenguaje de programación, *Csound* también dispone de diversos tipos de variables que podemos utilizar. Una *variable* es un dato que puede modificar su valor a lo largo de la ejecución del programa. El tipo de la variable vendrá determinado por su primera letra (prefijo), que puede ser: `g`, `i`, `k` o `a`.

Además, las variables se pueden clasificar en globales y locales. Una variable *global* es aquella que es común a toda la orquesta, y por consiguiente, puede ser utilizada por cualquier instrumento de ésta. Una variable *local* sería aquella creada dentro del cuerpo de un instrumento, por lo que solo puede ser utilizada por dicho instrumento, siendo invisible para el resto.

Los tipos de variables que encontramos en *Csound* son los siguientes:

Variables locales

- Tipo *nota*: Llevan el prefijo `i`. Toman valor al principio de cada nota que toca el instrumento en el que está definida. Ejemplo de uso: para establecer la frecuencia de un oscilador (`ifrec`).
- Tipo *control*: Contienen el prefijo `k`. Toman valor tantas veces por segundo como indica la frecuencia de control `kr`. Cada valor nuevo calculado sustituye al anterior. Ejemplo de uso: a la hora de implementar una envolvente de tipo ADSR que controle la amplitud de un oscilador (`kenv`).
- Tipo *audio*: Presentan el prefijo `a`. Toman valor tantas veces por segundo como indica la frecuencia de muestreo `sr`. Son vectores (o *arrays*) y cada valor se almacena a continuación del anterior. Ejemplo de uso: para almacenar la señal que genera un oscilador (`asalida`).

Variables globales

- Llevan el prefijo `g` más cualquiera de los prefijos anteriores, seguido del nombre de la variable. Por tanto, una variable global puede ser de tipo nota, control o audio. Es necesario declarar las variables globales al principio de la orquesta (normalmente en la cabecera) y toman valor al principio de la compilación. Ejemplos de éstas variables podrían ser: `gitabla`, `gkenv` o `gabus`.

3.2.4. Operadores

Un *operador* maneja uno o varios datos (operandos) para realizar un determinado cálculo con ellos. Los operadores en *Csound* pueden ser aritméti-

cos, relacionales o lógicos y presentan una sintaxis inspirada en el lenguaje C. Los principales operadores de *Csound* se muestran en la siguiente tabla:

Operadores aritméticos	
`+`	Suma
`-`	Resta
`*`	Producto
`/`	Cociente
`^`	Potencia
`%`	Resto de la división
Operadores relacionales	
`==`	Comparación
`<`	Menor que
`<=`	Menor o igual que
`>`	Mayor que
`>=`	Mayor o igual que
`!=`	Distinto de
Operadores lógicos	
`&&`	Conjunción (AND)
`||`	Disyunción (OR)
Operadores de asignación	
`=`	Asignación

Tabla 3.1: Operadores principales de *Csound*.

3.3. Fichero de partitura (.sco)

Desde el fichero de partitura podremos controlar los instrumentos que hayamos definido en la orquesta, crear las tablas de onda que utilizarán los *opcodes* generadores de señal y controlar el tempo de la secuencia sonora. La Fig. 3.7 muestra un fichero genérico de partitura en *Csound*. Dentro de dicho fichero, podemos observar que existen tres tipo de sentencias: `t`, `f` e `i`.

Fichero de partitura (`.sco`)

```
; Control del tempo.
t 0 90 10 95 20 100 30 120

; Generación de tablas de onda.
f 101 0 1024 10 1  ; Tabla 101 (sinusoidal).
   ...
; Control de los instrumentos.
i 1 0 5 25000 500   ; Instrumento 1.
i 2 0 5 10000 1000  ; Instrumento 2.
   ...
e  ; Fin de la partitura.
```

Figura 3.7: Fichero genérico de partitura en *Csound*.

3.3.1. Control del tempo

Las sentencias tipo `t` son instrucciones que nos permitirán establecer la evolución del tempo a lo largo del tiempo[4]. La forma característica de una sentencia `t` es la siguiente:

```
t   0 T1 tA T2 tB T3 tC ...
```

Figura 3.8: Sintaxis de una sentencia tipo `t` en *Csound*.

La sentencia comienza con la letra `t` seguida de los siguiente parámetros:

- `T1`, `T2`, `T3 ...` : Valores de tempo medidos en BPM (*Beats* Por Minuto).
- `0`, `tA`, `tB`, `tC ...` : Valores de tiempo medidos en *beats* o pulsaciones.

Lo que sucedería al escribir ésta sentencia es que en el tiempo `0` (*beat* `0`) se establecería el tempo `T1`. Al llegar al tiempo `tA`, el tempo habrá pasado de manera progresiva de `T1` al nuevo valor `T2`. Seguidamente, cuando se alcance el tiempo `tB`, el nuevo tempo sería de `T3`, así sucesivamente.

Por tanto, no estamos haciendo otra cosa sino especificar la curva del tempo que seguirá la secuencia sonora en función del tiempo:

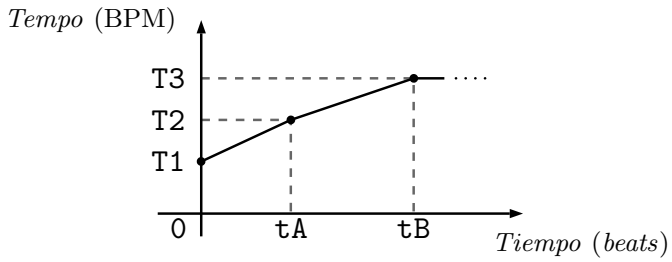


Figura 3.9: Curva de tempo generada mediante una sentencia de tipo `t`.

Nótese que tanto los parámetros de tempo como los de tiempo siempre tienen que ser mayores o iguales a cero, es decir `Ti>=0` y `ti>=0`. Además de ello, cualquier tiempo `ti` tiene que ser mayor o igual que el anterior `t(i-1)`, con lo que: `ti>=t(i-1)`.

En el caso en que deseáramos un tempo constante para toda la pieza sonora, solo tendríamos que especificar el tiempo inicial (`t=0`) y a continuación el tempo deseado. Por ejemplo, para un tempo constante de 92 BPM, la sentencia seria: `t 0 92`.

[4]Recordemos que el *tempo* es la velocidad a la que se ejecuta una pieza musical. Se mide en BPM (*Beats Per Minute*).

Si no escribiéramos la sentencia tipo `t`, *Csound* ajustaría automáticamente el tempo a 60 BPM (tempo por defecto).

3.3.2. Generación de tablas de onda

Para generar las tablas de onda necesarias para nuestros instrumentos haremos uso de las sentencias tipo `f`. Ésta sentencia provoca que una rutina GEN calcule los valores para un ciclo de onda (función matemática), que se almacenarán en una tabla. Una *rutina* GEN determinará, por tanto, cómo se crea dicha función. En *Csound* se pueden encontrar más de 20 rutinas GEN diferentes.

La estructura que sigue una sentencia tipo `f` sería la que se muestra a continuación: Donde:

```
f   n°tabla  t_creación  longitud  n°GEN  parámetros_GEN
```

Figura 3.10: Sintaxis de una sentencia tipo `f` en *Csound*.

- `n°tabla`: Establece el número con el que identificaremos la tabla almacenada.
- `t_creación`: Tiempo en el que se procederá a la creación de la tabla (medido en *beats*).
- `longitud`: Longitud de la tabla (nº de puntos de la función creada). Debe ser potencia de 2, siendo el tamaño máximo $2^{24} = 16777216$ puntos.
- `n°GEN`: Indica el número de la rutina GEN mediante la cual se creará la función almacenada en la tabla.
- `parámetros_GEN`: Parámetros necesarios que utilizará la rutina GEN especificada. Los parámetros cambiarán dependiendo de la rutina escogida.

Ejemplo: Crear una tabla de onda sinusoidal en *Csound* que pueda ser utilizada por un instrumento de la orquesta.

Por ejemplo, si definimos la tabla `f 101 0 32 10 1`, su número de identificación será `101`, por lo que si deseamos utilizarla desde algún instrumento de la orquesta, tendremos que especificar en el argumento correspondiente del *opcode* el texto `101`.

Por otra parte, nos indica que la estamos creando desde el inicio de la secuencia sonora, ya que `t_creación = 0` *beats*. Normalmente, las tablas se suelen crear para el *beat* 0.

En éste caso, la longitud de la tabla (que hará referencia al número de puntos de la función sinusoidal) será de 32 puntos, que como podemos comprobar, es potencia de 2 ($32 = 2^5$). Se ha tomado un número reducido de puntos para ilustrar éste ejemplo; no obstante, en la práctica utilizaremos valores de 1024 puntos en adelante.

Para crear la onda sinusoidal, hemos optado por la rutina GEN nº10. Ésta rutina es capaz de generar formas de onda a partir de una suma ponderada de armónicos sinusoidales. Los parámetros que se introducen para ésta rutina son las amplitudes normalizadas (A_{f_0}, A_{2f_0}, A_{3f_0},...) de los armónicos (f_0, $2f_0$, $3f_0$,...) que formarán la señal. Nótese que solo hemos introducido el valor `1` como parámetro. El motivo es que solo necesitamos el primer armónico con amplitud máxima 1, por lo que la función resultante de la tabla de onda será la función sinusoidal: $\text{sen}\,(2\pi f_0 t)$. En el caso en el que hubiéramos escrito la sentencia `f 101 0 32 10 1 0.5 0.3`, estaríamos creando la onda: $\text{sen}\,(2\pi f_0 t) + 0,5 \cdot \text{sen}\,(2\pi \cdot 2f_0 t) + 0,3 \cdot \text{sen}\,(2\pi \cdot 3f_0 t)$. Si deseáramos asignar amplitudes normalizadas a los parciales en forma de fracción, tendríamos que escribirla entre corchetes, por ejemplo: `f 101 0 32 10 1 [1/2] [1/3]`.

La frecuencia fundamental f_0 se definirá en el *opcode* que utilice la tabla creada, como ya se vió en la sección anterior.

La siguiente figura ilustra la representación de la tabla que acabamos de crear:

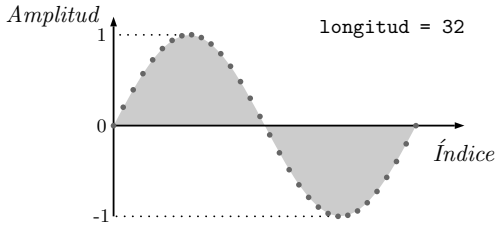


Figura 3.11: Representación de la tabla sinusoidal del ejemplo (`101`).

3.3.3. Ejecución de notas

Para que un instrumento de la orquesta sepa qué nota debe tocar y cuándo debe ejecutarla, tendremos que especificárselo mediante una senten-

cia tipo `i`. La sintaxis que sigue la sentencia tipo `i` aparece en la Fig. 3.12.

```
i    n°instrumento  t_inicio  duración  otros_parámetros
```

Figura 3.12: Sintaxis de una sentencia tipo `i` en *Csound*.

En ella, podemos distinguir los siguientes argumentos de entrada:

- `n°instrumento`: Número o identificador del instrumento que deseamos que ejecute la nota.
- `t_inicio`: Tiempo musical o *beat* en el que queremos que comience a reproducirse la nota.
- `duración`: Duración de la nota medida en *beats*.
- `otros_parámetros`: Se trata de varios parámetros que permitirán modificar características del instrumento de la orquesta al que hacemos referencia (amplitud, frecuencia, etc).

Por ejemplo, si deseáramos tocar una nota de un segundo con el instrumento que definimos en el apartado correspondiente a la orquesta (`instr 1`), tendríamos primero que establecer el tempo a 60 BPM, de modo que cada tiempo musical equivalga a 1 segundo. A continuación, solo quedaría indicar que comenzaremos a reproducir la nota desde el tiempo inicial (`0`) y la duración será de 1 tiempo, es decir: `i 1 0 1`.

Existen también otros caracteres especiales que podemos introducir en una sentencia `i` para simplificar el trabajo, sobretodo cuando tenemos que escribir muchas notas:

- Carácter "`.`": Si colocamos éste símbolo en algún parámetro de una sentencia `i`, se le asignará el mismo parámetro de la sentencia `i` anterior.
- Carácter "`<`": Si utilizamos éste carácter en algún parámetro de una sentencia `i`, se sustituirá por el valor interpolado de ese mismo parámetro de las líneas anteriores y posteriores.
- Carácter "`+`": Sustituye el tiempo de inicio de una sentencia tipo `i` por la suma de la duración y el tiempo de inicio de la sentencia `i` anterior.

Por último, *Csound* también presenta algunos comandos importantes que podemos utilizar en la partitura:

- `e`: Fin de partitura.
- `s`: Fin de sección (resetea la cuenta de tiempos).
- Etc...

3.4. Comunicación entre partitura y orquesta

Para posibilitar la comunicación entre la partitura y la orquesta, tenemos que hacer uso de los *campos* `p`. En realidad, todos los parámetros de entrada de una sentencia tipo `i` se denominan campos `p` y permitirán mandar información a cualquier instrumento definido en la orquesta. Por tanto, la estructura que ve *Csound* para la sentencia `i` sería la siguiente:

```
i   p1  p2  p3  p4  p5  [p6...pN]
```

Figura 3.13: Estructura de una sentencia tipo `i` en función de los campos `p`.

Los campos `p1`, `p2` y `p3` ya sabemos que se corresponden con los parámetros ya vistos: `nºinstrumento`, `t_inicio` y `duración`, respectivamente.

Ahora bien, los campos `p4` y `p5` hacen referencia normalmente a la amplitud (en PCM) y frecuencia (expresada en Hz) del instrumento cuyo identificador sea `p1`. No obstante, podría intercambiarse el orden.

Por otro lado, también pueden existir más parámetros `p` según la implementación del instrumento (`[p6...pN]`). Imaginemos, por ejemplo, que nuestro instrumento presentase una envolvente y deseáramos modificar el tiempo de ataque, especificando su valor desde la partitura. Éste control tendríamos que hacerlo mediante un campo `p`.

En definitiva, los campos `p` pueden entenderse como parámetros pertenecientes a algún instrumento de la orquesta cuyo valor lo introducimos desde la partitura. En cualquier caso, tenemos que indicar dentro del cuerpo del instrumento a qué campo `p` se refiere la variable de interés, por ejemplo, si es la amplitud: `iamp = p4`. Finalmente, habría que darle el valor a `p4` desde el fichero de partitura.

Veamos un ejemplo para entender mejor lo comentado:

Ejemplo: Implementar en *Csound* un oscilador sinusoidal con envolvente ADSR. Los parámetros característicos, tanto de la envolvente como del oscilador, deberán ser modificables desde el fichero de partitura empleando los campos `p`. Utilizar el instrumento creado para reproducir una nota LA_4 (440 Hz) de 3 segundos de duración.

Una posible solución podría ser la que aparece en la Fig. 3.14.

En el fichero de orquesta, empezamos declarando las variables globales reservadas. El número de canales de salida será `1` (`nchnls = 1`) dado

Fichero de orquesta (`.orc`)

```
;Variables globales reservadas.
nchnls = 1 ;Mono.
sr = 44100 ;Frecuencia de muestreo.
kr = 4410 ;Frecuencia de control.
ksmps = 10 ;Número de muestras por periodo de control.

;Oscilador sinusoidal con envolvente ADSR y parámetros
;ajustables desde la partitura.
instr 1
      iamp = p4 ;Amplitud.
      ifrec = p5 ;Frecuencia.
      iata = p6 ;Tiempo de ataque.
      idec = p7 ;Tiempo de decaimiento.
      isus = p8 ;Amplitud de sostenimiento.
      irel = p9 ;Tiempo de relajación.
      itabla = 101;Tabla de onda sinusoidal.
      kenv adsr iamp, idec, isus, irel ;Envolvente ADSR.
      asal oscil kenv*iamp, ifrec, 101 ;Oscilador sinusoidal.
      out asal ;Sacamos la señal de salida.
endin
```

Fichero de partitura (`.sco`)

```
;Tabla de onda sinusoidal (GEN 10).
f 101 0 2048 10 1
;Uso del instrumento 1.
i 1 0 3 15000 440 0.3 0.15 1 0.2
;Fin de partitura.
e
```

Figura 3.14: Fichero de orquesta y partitura para el ejemplo de oscilador sinusoidal con envolvente (incorporando campos `p`).

que la señal será monofónica. La frecuencia de muestreo se asigna a `44100` Hz (`sr = 44100`), mientras que la frecuencia de control la fijamos en el valor `4410` (`kr = 4410`). Como el número de muestras calculadas con los mismos valores de control (`ksmps`) es el cociente entre las dos variables anteriores, le asignamos el valor `10`, ya que `44100/4410 = 10`.

A continuación, definimos el instrumento `instr 1` que representará el oscilador sinusoidal. Dentro de éste, definimos diversas variables de tipo nota, correspondientes a la amplitud y frecuencia del oscilador (`iamp` e `ifrec`) y a los parámetros de la envolvente ADSR: tiempo de ataque, tiempo de decaimiento, amplitud de sostenimiento y tiempo de relajación (`iata`, `idec`, `isus` e `irel`, respectivamente). Como podemos observar, todas las variables mencionadas se han asignado a campos `p`, de modo que se les pueda dar valor desde la partitura. La última variable que creamos será `itabla`, que almacenará el número de tabla que crearemos en la partitura (`itabla = 101`).

En éste punto, creamos la envolvente ADSR a partir del *opcode* `adsr`. Ésta función devuelve una señal envolvente de tipo audio o control (`kenv`) a partir de los parámetros de entrada de los tiempos de las distintas fases (medidos en segundos) y la amplitud de sostenimiento. En el caso que nos ocupa, creamos la envolvente de tipo control `kenv` a partir de las variables declaradas, puesto que controlará la amplitud del oscilador que posteriormente definiremos: `kenv adsr iata idec isus irel`.

Respecto a la implementación del oscilador, haremos uso del *opcode* `oscil`, que generará la señal de salida `asal`. Para poder "conectar"la envolvente de amplitud `kenv` al oscilador, deberemos introducirle como primer parámetro de entrada (el de amplitud) el producto `iamp*kenv`, por lo que la amplitud final ya no será constante, sino variable en el tiempo. Por último, introducimos el parámetro correspondiente a la frecuencia y al número de la tabla de onda: `ifrec` e `itabla`.

Finalmente, sacamos la señal de salida generada a la tarjeta de sonido mediante el *opcode* `out`.

En el fichero de partitura, podemos ver que no hemos establecido ninguna instrucción tipo `t` para indicar el tempo de la secuencia. Ésto es debido a que *Csound* fija por defecto un tempo constante de 60 BPM. Por consiguiente, cuando no pongamos ningún tempo *Csound* interpretará la instrucción: `t 0 60`.

Proseguimos creando la tabla de onda sinusoidal de `2048` puntos a partir de la rutina GEN `10`, cuyo identificador o número de tabla será `101`. Nótese que solo se ha utilizado el primer armónico con amplitud normalizada `1`.

Si avanzamos en el código, observamos una sentencia `i` para poder utilizar el instrumento `instr 1`. Ésta sentencia recibe un `1` como número de instrumento (`p1 = 1`), inicia la nota desde el tiempo musical cero (`p2 = 0`) con duración `3` tiempos (`p3 = 3`), ya que a 60 BPM, 3 tiempos equivalen a 3 segundos. La amplitud `p4` que se mandará al oscilador del `instr 1` será de `15000` y la frecuencia `p5` de 440 Hz. Por último, introducimos un ataque de `0.3` segundos (p6 = 0.3), decaimiento de `0.15` segundos (p7 = 0.15), amplitud de sostenimiento `1 p8 = 1` y tiempo de relajación `0.2` segundos (`p9 = 0.2`). Acabamos, por último, con el comando de fin de partitura (`e`). El diagrama de bloques que ilustra el sintetizador creado quedaría de la siguiente manera:

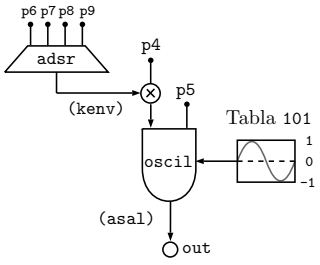


Figura 3.15: Diagrama de bloques para el ejemplo de oscilador sinusoidal con envolvente (incorporando campos `p`).

Capítulo 4

Diseño del sintetizador

El siguiente capítulo se centrará en abordar el diseño de un sintetizador FM que sea capaz de trabajar en tiempo real y pueda ser controlado por un teclado externo vía MIDI. El sistema deberá contar no solo con las funciones básicas de un sintetizador FM, sino con características más avanzadas que permitan explotar el potencial de la técnica de síntesis FM simple, facilitando a la par la docencia de la misma. Partiremos de una versión inicial o prototipo y continuaremos mejorando el diseño, comprobando en todo momento mediante pruebas específicas que las nuevas características añadidas responden correctamente. El *software* que nos servirá como entorno de desarrollo será *QuteCsound*, por lo que nos permitirá compilar el código fuente y diseñar la interfaz del sintetizador. Veremos también los problemas encontrados y qué soluciones se han adoptado.

4.1. QuteCsound

QuteCsound es un potente e intuitivo *front-end* (interfaz) para *Csound* con editor de código que permite trabajar en tiempo real, posibilitando a través de objetos gráficos (denominados *widgets*) interactuar con el sintetizador implementado. Contiene además, muchas otras características interesantes como: eventos en vivo, consola de salida, posibilidad de grabación, sintaxis coloreada e incluso autocompletado para facilitar la programación al usuario. Éste entorno de desarrollo es multilenguaje, multiplataforma, gratuito y libre (bajo licencia GPLv3)[1].

Para el presente proyecto, utilizaremos *QuteCsound* 0.6.0 (*double* version), así como *Csound* 5.18 (*double version*). El término *double* hace referencia a que se utilizará una precisión en el procesado de 64 bits, lo cual aportará mayor calidad sonora a la señal de salida pero también

[1] *QuteCsound* ha sido desarrollado a partir de *Qt*, un entorno de desarrollo de C++ multiplataforma y actualmente gratuito, propiedad de la empresa *Digia*.

necesitará más tiempo en el procesado interno. También existe otra versión de *Csound*/*QuteCsound* tipo *float*, que trabaja con la mitad de precisión (32 bits). Una cuestión muy importante a la hora de la instalación es comprobar que tanto la versión de *Csound* como la de *QuteCsound* concuerden en precisión, es decir, no se debe mezclar una versión *float* con una *double*.

4.1.1. Interfaz principal

Una vez instalado instalados ambos programas, ejecutamos *QuteCsound* y encontramos la siguiente interfaz:

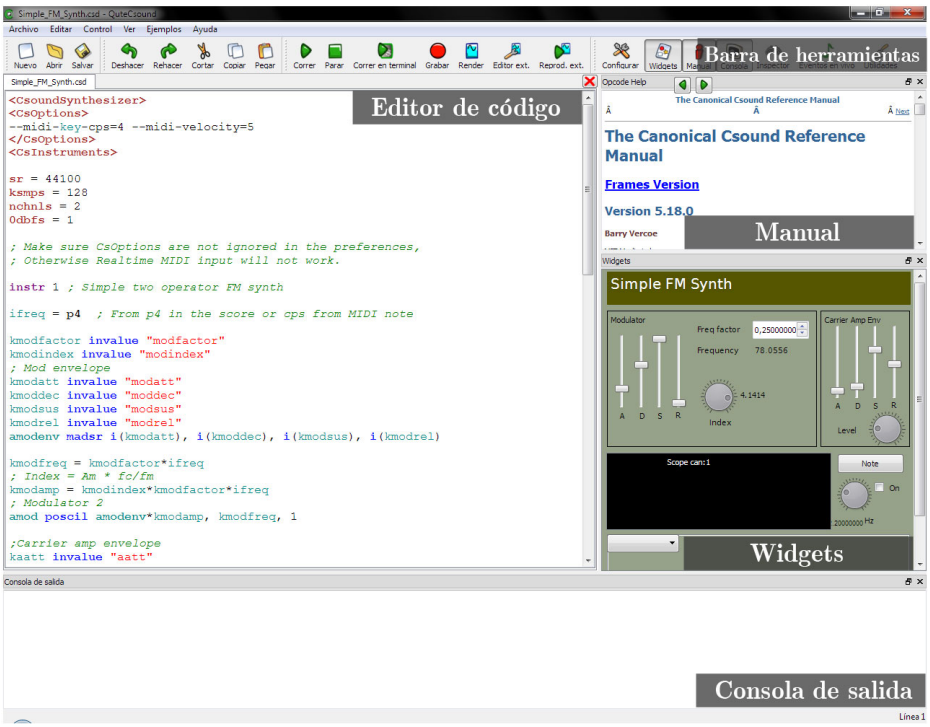


Figura 4.1: Interfaz del software *QuteCsound* con los principales paneles señalados.

Si observamos la figura anterior, podremos distinguir varios paneles o subventanas que emplearemos a lo largo del proyecto:

- **Barra de herramientas**: Está formada por numerosos botones que implementan accesos directos a opciones del menú principal. Ejecutar el programa (botón *Correr*), pausar su ejecución (botón *Parar*), grabar la secuencia sonora (botón *Grabar*) o guardar el fichero (botón *Salvar*) serían solo algunas de las opciones que podemos seleccionar desde la barra de herramientas. Debajo de la barra de herramientas también podemos encontrar diversas pestañas que nos permiten acceder a un fichero u otro, en caso de que tengamos varios archivos abiertos.

- **Editor de código**: Es el espacio en el que podemos escribir el código fuente del proyecto. Presenta sintaxis coloreada, esto es, que según el tipo de variable o sentencia que escribamos, el editor cambiará automáticamente su color para facilitar la labor de programación. También tiene autocompletado de sintaxis, por lo que al empezar a escribir alguna palabra reservada de *Csound*, el editor nos ofrecerá una lista con las palabras más probables que se ajustan con la nuestra, lo cual simplifica la redacción del código.

- **Widgets**: Es un panel dedicado al conjunto de *widgets* que forman la interfaz gráfica del sintetizador que estamos implementando. Dicha interfaz permitirá al usuario interactuar en tiempo real con el sintetizador creado. *QuteCsound* dispone de 14 *widgets* mediante los que podremos personalizar la interfaz y, además, son editables gráficamente, por lo que se reduce la extensión del código. Más adelante los trataremos en mayor profundidad.

- **Manual**: Es un manual de ayuda para el usuario con formato HTML, en el que podemos consultar información sobre los *opcodes*, rutinas GEN...etc. Si escribimos un *opcode* en el editor de código, seleccionamos su texto y pulsamos el comando `Shift+F1`, nos remitirá a la sección del manual donde aparecerá información relativa al *opcode* escrito.

- **Consola de salida**: Muestra los mensajes de salida de *Csound*. Puede resultar útil si queremos averiguar información de alguna variable en algún punto del programa (utilizando *opcodes* tipo `print`).

4.1.2. El fichero .csd

QuteCsound tiene la ventaja de trabajar con ficheros `.csd`, ya que éstos engloban el fichero de orquesta (`.orc`), el fichero de partitura (`.sco`) y todos los *widgets* que hayamos creado en el archivo. Los *widgets* almacenados presentan la particularidad de que su código resulta invisible para el usuario cuando abrimos el fichero `.csd` con *QuteCsound* (Fig. 4.2). Con ello, se evita introducir código que "ensucie" el proyecto, facilitando la labor al programador.

Puesto que tenemos la orquesta y la partitura en el mismo archivo, necesitamos algún identificador que le diga a *QuteCsound* qué parte de código se corresponde con cada una. Por ello, tenemos una sintaxis adicional en *QuteCsound* que debemos incorporar a la que ya conocíamos en *Csound*:

- `<CsoundSynthesizer>` y `</CsoundSynthesizer>`: Indican el principio y el final del archivo `.csd`, respectivamente (sin contar los *widgets*).

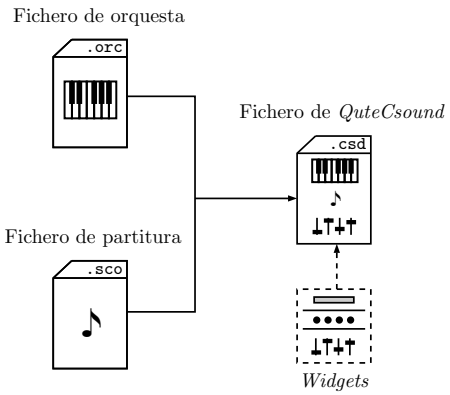


Figura 4.2: Diagrama básico de un archivo `.csd`.

Contendrá las secciones correspondientes a la orquesta, partitura y otras opciones de *Csound*.

- `<CsOptions>` y `</CsOptions>`: Entre éstos dos identificadores podemos programar distintas opciones de *Csound*, activando o desactivando líneas de comando denominadas *flags*. Existen *flags* para modificar el archivo de audio de salida, mostrar mensajes MIDI en tiempo real a través de la consola, configurar las entradas/salidas de la tarjeta de audio, etc.
- `<CsInstruments>` y `</CsInstruments>`: Delimitan la zona del código donde implementaremos la orquesta.
- `<CsScore>` y `</CsScore>`: Indican el comienzo y el fin de la partitura.

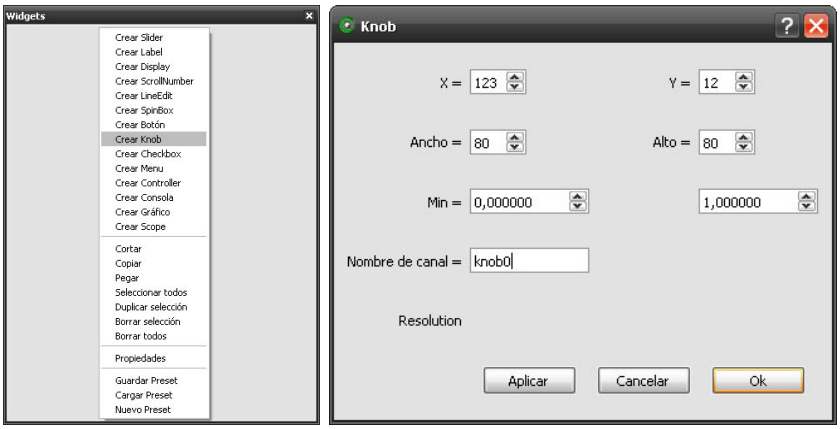
4.1.3. Widgets

Los *widgets* son elementos gráficos que nos permiten interactuar en tiempo real con el sintetizador que estamos programando. El conjunto de todos los *widgets* que tengamos guardados en el archivo `.csd` constituirían la interfaz gráfica (GUI, *Graphical Unit Interface*) del sintetizador. Ésta herramienta es característica de *QuteCsound* puesto que no forma parte de *Csound* por sí mismo. *QuteCsound* es el responsable de añadir el código de manera oculta en el archivo `.csd`, justo a continuación de la etiqueta `</CsoundSynthesizer>`.

QuteCsound nos permite seleccionar un total de 14 tipos distintos de *widgets*, con los que podemos personalizar nuestro sintetizador, a fin de poder cubrir las necesidades del programador. Existen *widgets* encargados en salida de texto (*label* y *display*), entrada de texto (*lineedit*), entrada

numérica (*slider*, *knob*, *scrollnumber* o *spinbox*), botones (*button*), interruptores (*checkbox*) o incluso menús desplegables (*menu*) y paneles de representación gráfica (*graph* y *scope*), además de otros.

Para empezar a crear un *widget*, debemos tener activado el botón *Widgets* de la barra de herramientas, de modo que aparezca el panel de *widgets*. Si pulsamos sobre éste panel con el botón derecho del ratón[2], aparecerá una lista seleccionable de todos los *widgets* que podemos emplear (Fig. 4.3 (a)). Todo *widget* contiene, a su vez, un panel de propiedades modificables que dependerán del tipo de *widget* creado (Fig. 4.3 (b)).



(a) (b)

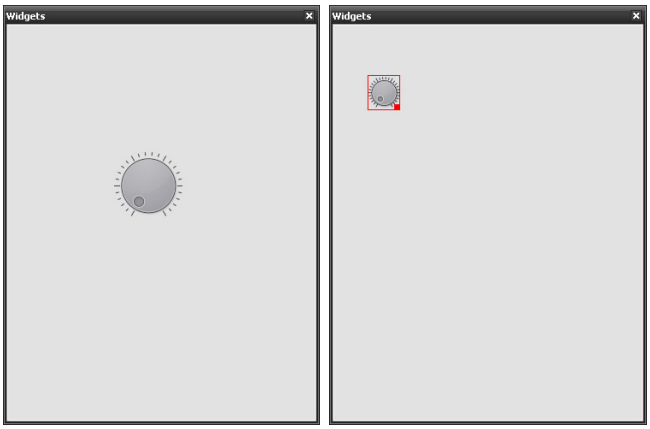
Figura 4.3: (a) Lista de widgets seleccionables en *QuteCsound*. (b) Panel de propiedades de un *widget*.

Existen 3 propiedades comunes a todos los *widgets*: su posición (X, Y) en el panel, sus dimensiones (*ancho* y *alto*) y el nombre del canal o identificador del *widget*. El *nombre del canal* es una propiedad muy importante ya que permitirá intercambiar información entre el *widget* y el código del sintetizador. Más adelante veremos cómo hacerlo.

Respecto a las propiedades específicas de cada *widget*, podemos destacar, por ejemplo, la edición de colores de fondo o de texto (en el caso del *label* y *display*), la resolución numérica o escalón de una entrada numérica (para el *widget spinbox* y otros) o el estado numérico que tendrá un botón al presionarlo.

Una vez creado el *widget* y modificadas sus propiedades a nuestras exigencias, lo veremos situado en el panel de *widgets*, según se indica

[2] Aunque *QuteCsound* es un entorno de desarrollo multiplataforma, éste proyecto se ha desarrollado en sistema operativo *Windows*.



(a) (b)

Figura 4.4: (a) Ejemplo de *widget* creado (*knob*). (b) Modo edición del panel de *widgets*.

en la Fig. 4.4 (a). Por otro lado, existe una forma cómoda de modificar gráficamente las dimensiones y la posición del *widget*, sin necesidad de introducirlas desde el panel de propiedades: el modo *edición*.

Para acceder al modo edición desde el menú principal de *QuteCsound*, tendremos que entrar en *Editar* → *Modo de edición*; o bien a través del comando `Ctrl + E`. A continuación, veremos que aparece un marco o *bounding box* de color rojo alrededor de cada *widget* del panel (Fig. 4.4 (b)).

Si mantenemos pulsada la esquina inferior derecha (rellena) de cada marco y arrastramos el ratón, modificaremos las dimensiones del *widget*. En cambio, si pulsamos en cualquier otro punto del *widget* y arrastramos, podremos desplazarlo a la posición que nos interese.

Para salir del modo edición, podemos volver a pulsar `Ctrl + E` o acceder a *Editar* → *Modo de edición*.

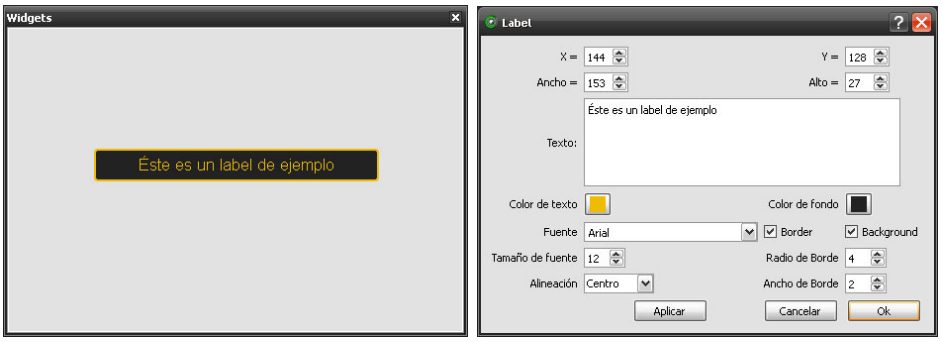
En el siguiente apartado, analizaremos muy brevemente los *widgets* más importantes utilizados en el proyecto.

4.1.3.1. Label

El *label* representa una etiqueta de texto a la que podemos añadirle formato: texto, fuente, color del texto, color del fondo, alineación, borde de texto...etc. El *label* es el único *widget* que no presenta nombre de canal. Ésto significa que no podemos interactuar con él desde el código del

sintetizador. Por tanto, el *label* se dedicará únicamente en *QuteCsound* al aspecto estético del interfaz.

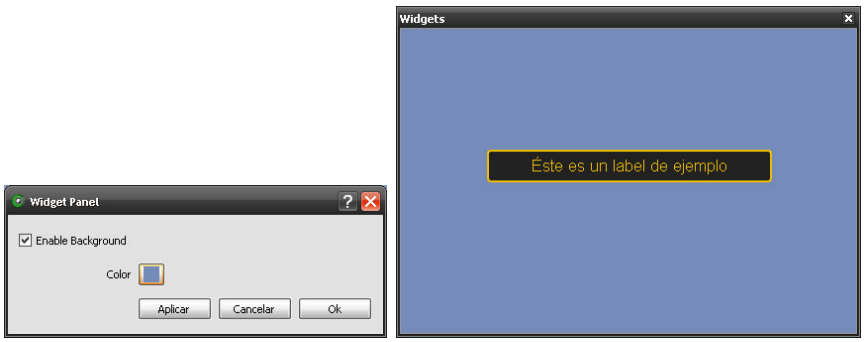
En éste ejemplo, se ha creado un *label* al cual se le ha añadido un color de fondo gris oscuro, color de texto naranja y un contorno levemente redondeado en sus esquinas:



(a) (b)

Figura 4.5: (a) Ejemplo de un *widget* tipo *label*. (b) Propiedades del *widget label*.

También tenemos la posibilidad de modificar el fondo o *background* a nuestro gusto. Para ello, pulsamos sobre el botón derecho del ratón en cualquier zona del panel que no contenga *widgets* y tras salir la lista, seleccionamos la opción *Propiedades* y a continuación, activamos la casilla *Enable Background*. Por último, seleccionamos el color de interés y pulsamos sobre los botones *Aplicar* y *Ok* (Fig. 4.6).



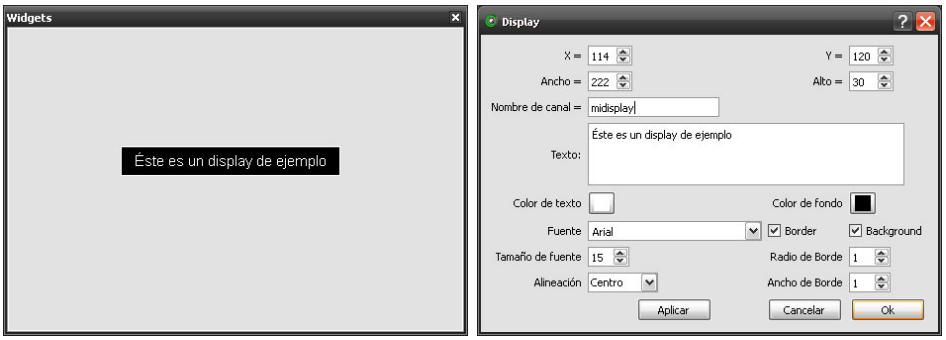
(a) (b)

Figura 4.6: (a) Menú de ajuste del color de fondo. (b) Resultado final tras aplicar el cambio de fondo.

4.1.3.2. Display

El *widget display* es idéntico al *label*, con la diferencia de que *display* sí que permite interacción con el código, pues contiene nombre de canal. Por tanto, el *display* nos permitirá sacar una cadena de texto (*string*) desde el programa del sintetizador, lo cual puede resultar útil en muchas ocasiones. Nosotros hemos aprovechado éste *widget*, por ejemplo, para implementar un detector de nota-ON MIDI.

En el siguiente ejemplo (Fig. 4.7), hemos creado un *widget* tipo *display* cuyo nombre de canal es *midisplay*, y aunque le hemos asignado un texto desde el panel de propiedades, el verdadero cometido del *display* es sacar el texto desde el programa.



(a) (b)

Figura 4.7: (a) Ejemplo de un *widget* tipo *display*. (b) Propiedades del *widget display*.

4.1.3.3. LineEdit

Éste *widget* posibilita la entrada de texto al código del programa. Además, el texto introducido es editable, por lo que podremos modificarlo en cualquier momento sin necesidad de entrar en el modo edición. No obstante, contiene las mismas propiedades de formato que un *display*. Un posible ejemplo de uso podría ser para introducir una nota musical, a fin de que el sintetizador la reproduzca (Fig. 4.8).

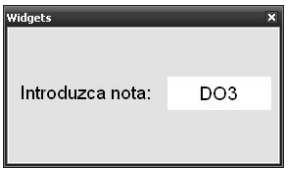
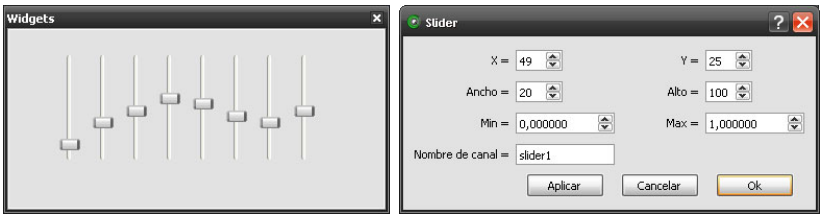


Figura 4.8: Ejemplo de un *widget* tipo *lineedit*.

4.1.3.4. Slider

Un *slider* es un *widget* con aspecto de *fader* o deslizador, que se utiliza para introducir un valor numérico al sintetizador. Dicho valor debe estar comprendido entre un valor mínimo y un valor máximo dependiendo de la posición del *slider*. En el panel de propiedades del *slider* podemos editar los campos *Min* y *Max*, correspondientes a los valores mencionados. Una posible desventaja del *slider* sería que no podemos editar su formato. Puede ser útil para controlar los niveles de las bandas de un ecualizador, tiempos y amplitud de las fases de una envolvente, etc (véase Fig. 4.9).



(a) (b)

Figura 4.9: (a) Ejemplo de varios *widgets* tipo *slider*. (b) Propiedades del *widget slider*.

4.1.3.5. Knob

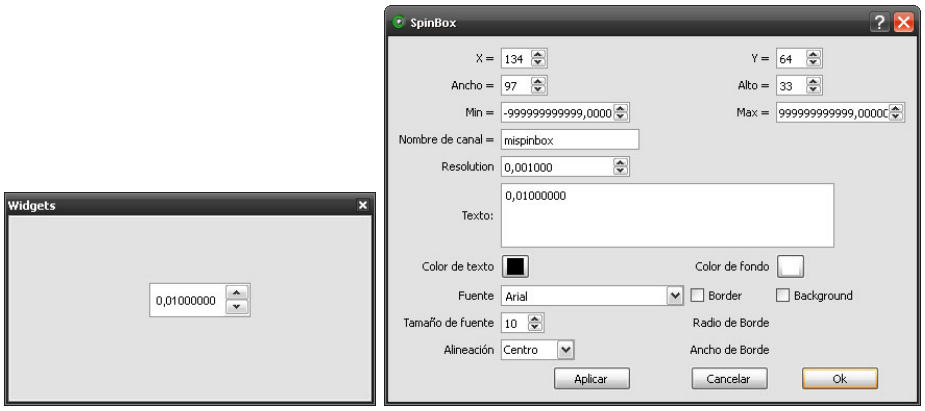
El *knob* se trata de otro *widget* especializado en entrada numérica (idéntico al *slider*), pero en ésta ocasión, presenta forma de potenciómetro en lugar de deslizador. Es una alternativa estética al *slider* que cubre la misma función, ocupando por lo general menos espacio que el anterior *widget*. En el proyecto, veremos que lo hemos empleado en multitud de ocasiones: para variar el índice de modulación, para modificar los parámetros de los efectos digitales, etc. En la Fig. 4.10 podemos ver el aspecto y las propiedades de éste *widget*:

(a) (b)

Figura 4.10: (a) Ejemplo de varios *widgets* tipo *knob*. (b) Propiedades del *widget knob*.

4.1.3.6. SpinBox

Un *spinbox* es un recuadro con formato editable que sirve también para introducir un dato numérico en el programa y permite visualizar, a diferencia de los dos *widgets* anteriores, el valor en el que nos encontramos. Éste valor lo podemos seleccionar a través de dos flechas adyacentes al recuadro. Otro aspecto interesante de éste *widget* consiste en que podemos establecer la resolución o escalón entre los valores máximo y mínimo, es decir, podemos decidir avanzar de unidad en unidad, de décima en décima, de milésima en milésima...etc.



(a) (b)

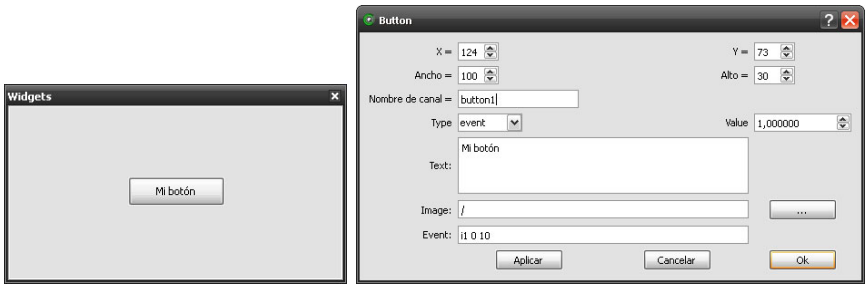
Figura 4.11: (a) Ejemplo de un *widget* tipo *spinbox*. (b) Propiedades del *widget spinbox*.

4.1.3.7. Button

Un *widget button* es un elemento que puede contener un texto o imagen de fondo y permite cambiar su estado numérico al pulsarlo, de modo que desde el programa se puede consultar su valor y programar alguna acción en caso de ser pulsado. El parámetro *value* permite asignar el valor que tendrá el botón al activarlo. Además, en *QuteCsound*, el *button* puede asociarse a una sentencia tipo `i`, para que al presionarlo, un determinado instrumento de la orquesta pueda ejecutar una nota.

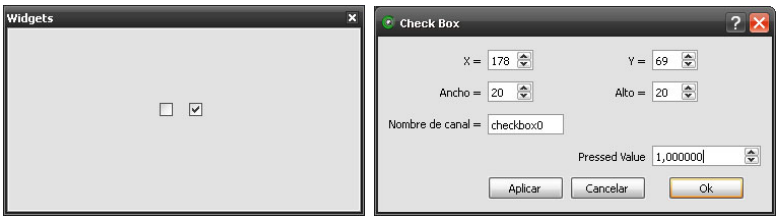
4.1.3.8. Checkbox

El *checkbox* es un *widget* con forma de casilla que puede activarse o desactivarse, como si se tratara de un interruptor. Normalmente, se suele establecer el valor `0` para desactivada y el valor `1` para activada, aunque éste último valor puede cambiarse desde el campo *pressed value*. El valor de la casilla se puede también consultar desde el programa del sintetizador.



(a) (b)

Figura 4.12: (a) Ejemplo de un *widget* tipo *button*. (b) Propiedades del *widget button*.



(a) (b)

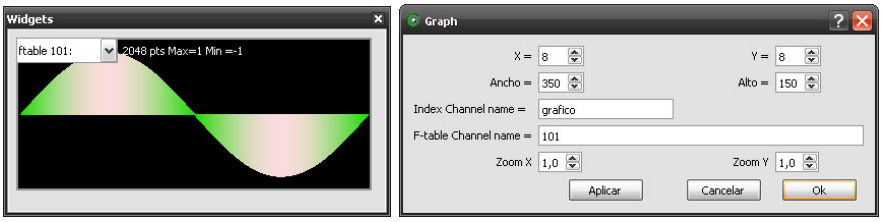
Figura 4.13: (a) Ejemplo de un *widget* tipo *checkbox*. (b) Propiedades del *widget checkbox*.

4.1.3.9. Graph

Permite visualizar la función almacenada en una tabla de onda o bien el espectro en tiempo real de alguna señal de nuestro sintetizador. Para ello, debemos especificar en el menú de propiedades a qué tabla nos dirigimos (mediante el parámetro *f-table channel name*) y si queremos alguna escala específica para los ejes de coordenadas X e Y (parámetros *zoom X* y *zoom Y*, respectivamente). Por último, se presenta un ejemplo en el que se ha asociado un *widget* tipo *graph* a la tabla de onda sinusoidal *101* creada en un ejemplo anterior (Fig. 4.14)

4.1.4. Interacción entre sintetizador y widgets

Acabamos de estudiar muy brevemente algunos de los *widgets* que ofrece *QuteCsound*, no obstante, los *widgets* por sí solos (salvo el *label*) no son de utilidad si no interactúan con el sintetizador. Ahora bien, ¿cómo podemos interactuar con ellos a través del código implementado? La respuesta a ésta pregunta reside en los *opcodes* `invalue` y `outvalue`.



(a) (b)

Figura 4.14: (a) Ejemplo de un *widget* tipo *graph*. (b) Propiedades del *widget graph*.

El *opcode* `invalue` nos permite leer o recibir datos de un determinado *widget*, y más concretamente de un canal, puesto que cualquier *widget* representa un canal identificado por su nombre de canal.

Por tanto, `invalue` recibe como parámetro de entrada, una única cadena de texto (escrita entre comillas) correspondiente al nombre de canal del *widget* del que queremos recibir información. Como parámetros de salida, podemos obtener una variable de control (tipo `k`) si se tratara de un *widget* de entrada numérica; o bien una cadena de texto (tipo `S`, de `string`) si fuera un *widget* de entrada de texto. Las dos posibles sintaxis de `invalue` serían las que siguen:

```
k_variable  invalue  "nombre_de_canal"

S_variable  invalue  "nombre_de_canal"
```

Figura 4.15: Sintaxis del *opcode* `invalue`.

Por otro lado, el *opcdode* `outvalue` realiza la operación contraria, es decir, nos permite enviar datos a un *widget* en particular. En éste caso, el *opcode* `outvalue` no devuelve ningún parámetro de salida[3], pero recibe el nombre de canal (entre comillas) del *widget* al que deseamos enviarle información y una variable de tipo control o *string*, según deseemos enviar un dato numérico o de texto. Su sintaxis es la que se muestra en la Fig. 4.16.

Un aspecto negativo tanto de `invalue` como `outvalue` es que consumen mucha CPU, según al tipo de *widget* que vayan dirigidos. En éste sentido, tendremos que optimizar su uso si programamos en un ordenador con "pocos requerimientos". Más adelante veremos como mejorar éste problema.

[3]En lenguajes de programación como C o C++, las funciones que no devuelven ningún parámetro de salida son conocidas como *procedimientos*.

```
outvalue  "nombre_de_canal", k_variable

outvalue  "nombre_de_canal", S_variable
```

Figura 4.16: Sintaxis del *opcode* `outvalue`.

Finalmente, podemos ver un diagrama muy simple de cómo se intercambia información entre el sintetizador (archivo `.csd`) y los *widgets*, a través de los *opcodes* mencionados:

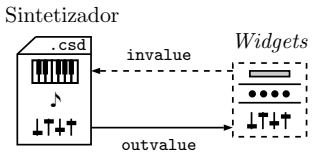


Figura 4.17: Diagrama de la función de los *opcodes* `invalue` y `outvalue`.

4.1.5. Presets

Otra opción muy interesante en *QuteCsound* es la de crear, guardar y cargar *presets*. Los *presets* son configuraciones de valores determinados, correspondientes a uno o más *widgets*.

Por ejemplo, podríamos crear un *slider* para controlar el volumen de la señal de salida del sintetizador. Podría interesarnos crear una configuración para volumen nulo (*silencio*), otra para mitad de volumen y una última para volumen máximo, almacenándolas posteriormente. De éste modo, no tendríamos que mover nosotros mismos el deslizador, sino cargar cualquiera de las distintas configuraciones o *presets* creados (Fig. 4.18).

Figura 4.18: *Slider* de ejemplo con 3 *presets* almacenados.

Para poder crear un *preset* en *QuteCsound*, debemos crear primero todos los *widgets* que queremos que formen parte de la configuración. A continuación,

debemos ajustar los parámetros que nos interesen de cada *widget* para formar el *preset* y después pulsar sobre el botón derecho del ratón, dentro del panel de *widgets*.

En la lista que aparece, podremos observar la opción *Nuevo Preset*. Si accedemos a ella, podremos ver un cuadro de diálogo que nos pedirá que introduzcamos el nombre del nuevo *preset*, así como su número. Finalmente, pulsaríamos sobre el botón *Ok* y ya habríamos finalizado la creación del *preset*, quedando éste almacenado.

En el ejemplo del *slider*, comenzaríamos creando dicho *widget* y algunos *label* para añadir información estética. Seguidamente, deslizaríamos el *slider* a la posición de interés (la mínima, en éste caso) y procederíamos a crear el *preset* que denominaremos `Silencio`, asignándole el número `0` (Fig. 4.19). Repitiendo el proceso para el resto de configuraciones, variando únicamente la posición del *slider* en cada caso a posición media y a la máxima, obtendríamos los *presets* de `Volumen medio` y `Volumen máximo`, respectivamente.

Figura 4.19: Creación y almacenamiento de un *preset* en *QuteCsound*.

Para poder cargar un *preset* guardado, volvemos a pulsar el botón derecho del ratón y seleccionamos la opción *Cargar Preset* de la lista. Cuando aparezca el cuadro de diálogo, hacemos doble *click* sobre el *preset* que deseamos cargar y pulsamos sobre el botón *Cerrar*. Retomando el ejemplo del *slider*, realizamos éste procedimiento para la carga del *preset* `Volumen medio` (véase Fig. 4.20).

Relacionados con los *presets*, en *QuteCsound* existen lo que se denominan *canales reservados*. Un canal reservado es básicamente un identificador (precedido del símbolo `"_"`) que implementa una función específica que afecta a *QuteCsound*.

En éste sentido, si creamos un *widget* y como nombre de canal le ponemos un canal reservado, estaremos vinculando a dicho *widget* la función que tenga implementada ese canal reservado.

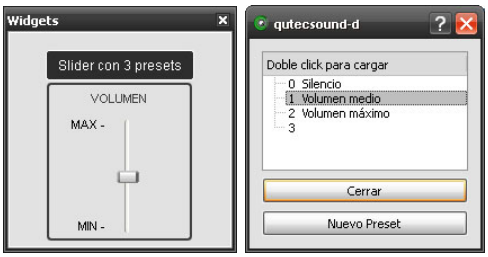


Figura 4.20: Carga de un *preset* en *QuteCsound*.

Existen 4 canales reservados en *QuteCsound* referentes al control de *presets*:

- `_GetPresetName`: Obtiene el nombre del *preset* que está cargado en ese momento.
- `_GetPresetNumber`: Recibe el número del *preset* cargado.
- `_SetPresetIndex`/`_SetPreset`: Establecen el número de *preset* que se va a cargar.

4.1.6. Ajustes de audio, MIDI y latencia

Para modificar cualquier ajuste referente al audio en *QuteCsound*, tenemos que acceder al menú *Editar* → *Configuración* o bien desde la barra de herramientas, pulsando sobre el botón *Configurar*. Si ahora seleccionamos la pestaña *Correr*, veremos la ventana que aparece en la Fig. 4.21.

En primer lugar, debemos seleccionar las entradas y salidas de audio que vayamos a utilizar. Para ello, en la sección *Reproducir en tiempo real*, seleccionamos como *módulo de audio RT*, la opción *portaudio* que aparece en la lista desplegable. A continuación, modificaríamos los campos correspondientes a *dispositivo de entrada* (`-i`) y *dispositivo de salida* (`-o`), seleccionando la entrada y/o salida de interés (pulsar sobre el botón que contiene puntos suspensivos).

En el presente proyecto no hemos utilizado entrada de audio, ya que el sintetizador no debe recibir ninguna señal externa.

Continuando en la sección *Reproducir en tiempo real*, a la derecha, podemos establecer el *módulo MIDI RT*, que ajustaremos en el valor *portmidi*. Los ajustes MIDI serán solo necesarios en caso de que necesitemos controlar el sintetizador a través de un teclado MIDI externo o deseemos enviar información desde el sintetizador a un dispositivo MIDI. Tanto el dispositivo de entrada (`-M`) como el de salida (`-Q`) se configurarían del mismo modo

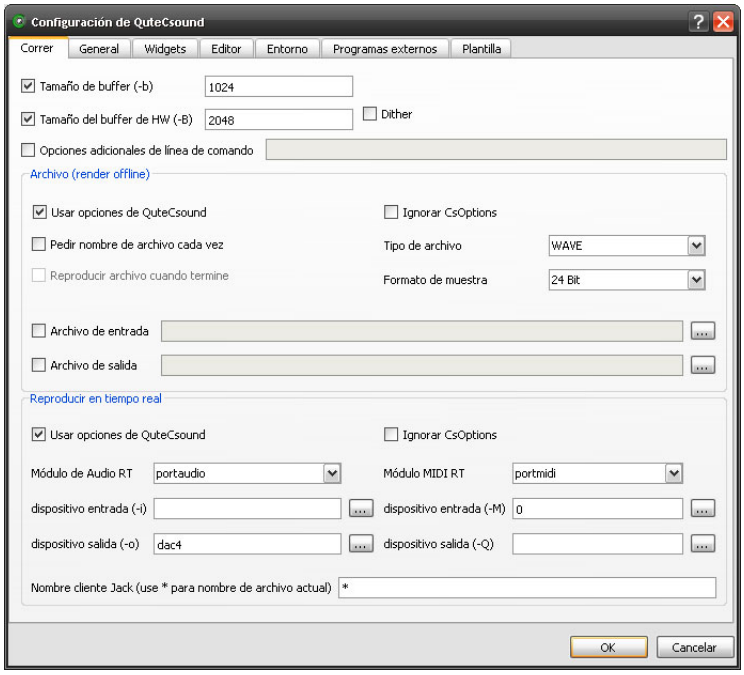


Figura 4.21: Pestaña *Correr* del menú de preferencias de *QuteCsound*.

que con el *módulo de audio RT*, seleccionando en cada caso el puerto MIDI al que tengamos conectado el dispositivo, bien sea de entrada o de salida.

Nótese que las entradas y salidas de audio y MIDI tienen asociadas *flags* específicas. Ésto se debe a que también podemos seleccionar las entradas y salidas necesarias desde el propio código, escribiendo los *flags* correspondientes dentro del bloque de código enmarcado por las etiquetas `<CsOptions>` y `</CsOptions>`. Nosotros nos hemos decantado por configurar ambos módulos desde preferencias, puesto que resulta más sencillo.

Finalmente, pasamos a los ajustes de latencia. Recordemos que la latencia representa el tiempo transcurrido entre la pulsación de un nota desde el controlador externo y la respuesta del sintetizador. Por tanto, al trabajar con audio en tiempo real, siempre nos interesará que la latencia sea mínima (a ser posible, inferior a 10 milisegundos), ya que en caso contrario, la interpretación sería prácticamente imposible.

Para optimizar la latencia en *QuteCsound*, debemos modificar dos campos que se denominan *tamaño de buffer* (`-b`) (*buffer* de *software*) y *tamaño del buffer de HW* (`-B`) (*buffer* de *hardware*). Valores muy bajos para ambos *buffers* permitirán reducir la latencia, pero corremos el riesgo de que aparezcan *clicks* en el sonido al utilizar el sintetizador. Por otro lado,

valores muy elevados harán que el sonido no se interrumpa, pero tendremos más latencia.
En éste sentido, será necesario conseguir un equilibrio en el que tengamos poca latencia y un sonido libre de chasquidos. Para lograrlo, debemos colocar valores que sean potencia de 2 en ambos *buffers*.

Normalmente, suelen conseguirse buenos resultados dando al *buffer* de *hardware* el doble de valor que al *buffer* de *software*. En nuestro caso, hemos obtenido buena latencia empleando la relación de valores `b:B = 256:512`, pero en caso de que se sigan oyendo chasquidos, podemos doblar ésta relación a `b:B = 512:1024`.

Otra herramienta que podemos utilizar, si no conseguimos una buena latencia modificando los *buffers* mencionados, es instalar ASIO4ALL.

ASIO4ALL es un *driver* ASIO[4] universal WDM (*Windows Driver Model*) de baja latencia, que no solo corrige los problemas de latencia, sino que ayuda a mejorar la calidad del sonido reduciendo los posibles chasquidos (Fig. 4.22). Es muy recomendable utilizar éste *driver* si disponemos de un *hardware* de audio de gama baja. Para utilizar ASIO4ALL con *QuteCsound*, tenemos que seleccionarlo como dispositivo de salida de audio (Fig. 4.23).

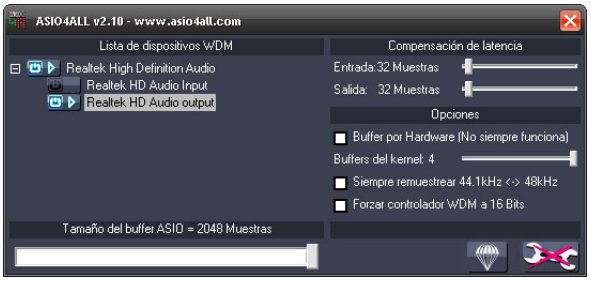


Figura 4.22: Interfaz del *driver* ASIO4ALL (versión 2.10).

4.2. Versión inicial del sintetizador

Ya tenemos algunas nociones sobre el entorno de desarrollo en el que vamos a trabajar, por lo que procederemos al diseño de la versión inicial o prototipo del sintetizador FM. En el capítulo 2 pudimos observar que un sintetizador que trabaje con FM simple deberá estar formado por dos osciladores: el portador y el modulador.

[4] *Audio Stream Input/Output*. Se trata de un estándar de audio digital para ordenador creado por la empresa *Steinberg*, que otorga baja latencia y una interfaz de alta fidelidad entre el *software* y la tarjeta de sonido.

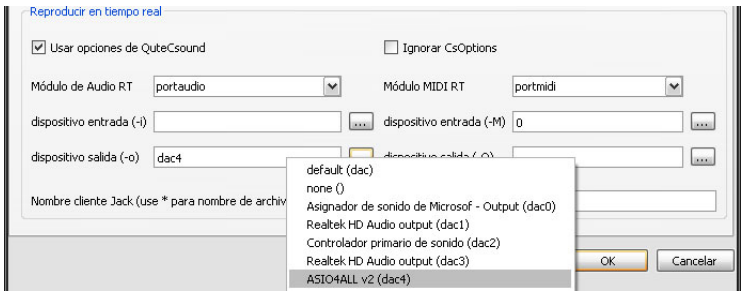


Figura 4.23: Configuración de ASIO4ALL en *QuteCsound*.

También estudiamos que los principales parámetros que representan a cada oscilador son su amplitud y frecuencia. En *Csound* también tendremos que tener otro parámetro en cuenta: el tipo de tabla de onda (sinusoidal, cuadrada...etc). En definitiva, qué tipo de señal generará el oscilador.

Otros parámetros decisivos en síntesis FM son el índice de modulación (para controlar la generación de parciales) y la relación $c : m$ (para gobernar la armonicidad del espectro).

A tales efectos y pensando en la aplicación docente, hemos propuesto controlar únicamente 3 parámetros para ambos osciladores: el índice de modulación, el factor de frecuencia de portadora (c) y el factor de frecuencia de moduladora (m). Con ello, conseguiremos simplificar el control del sintetizador y facilitar el diseño de los instrumentos. Escogeremos, inicialmente, una tabla de onda sinusoidal para ambos osciladores.

Además, cada oscilador deberá tener su propia envolvente ADSR ajustable, para ofrecer mayor libertad en el diseño sonoro y se pueda comprobar el efecto que produce la modificación de sus parámetros. La envolventes podrán ser activadas o desactivadas según la decisión del usuario.

Como conclusión, el sintetizador FM inicial deberá contar con las siguientes funciones:

- Control del índice de modulación, así como de los parámetros de la relación $c : m$.
- Envolventes ADSR ajustables para cada señal (portadora y moduladora) con posibilidad de encendido y apagado.
- Formas de onda de tipo sinusoidal para ambos osciladores.

Todos los controles se implementarán a través de los distintos *widgets* ya vistos que ofrece *QuteCsound*.

4.2.1. Diagrama de bloques del prototipo

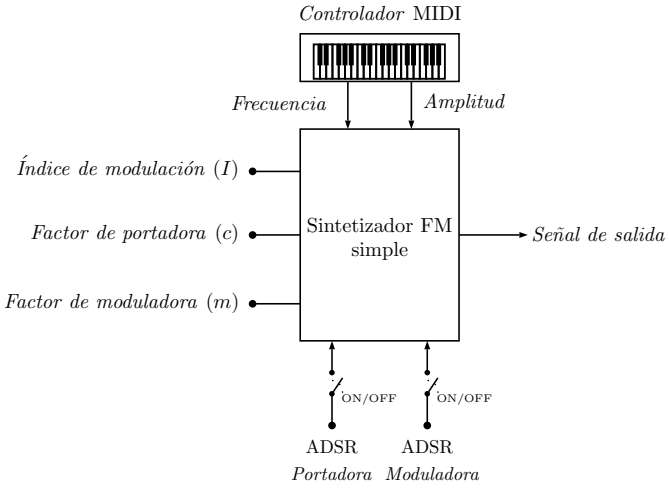


Figura 4.24: Diagrama de bloques simplificado de la versión inicial del sintetizador FM.

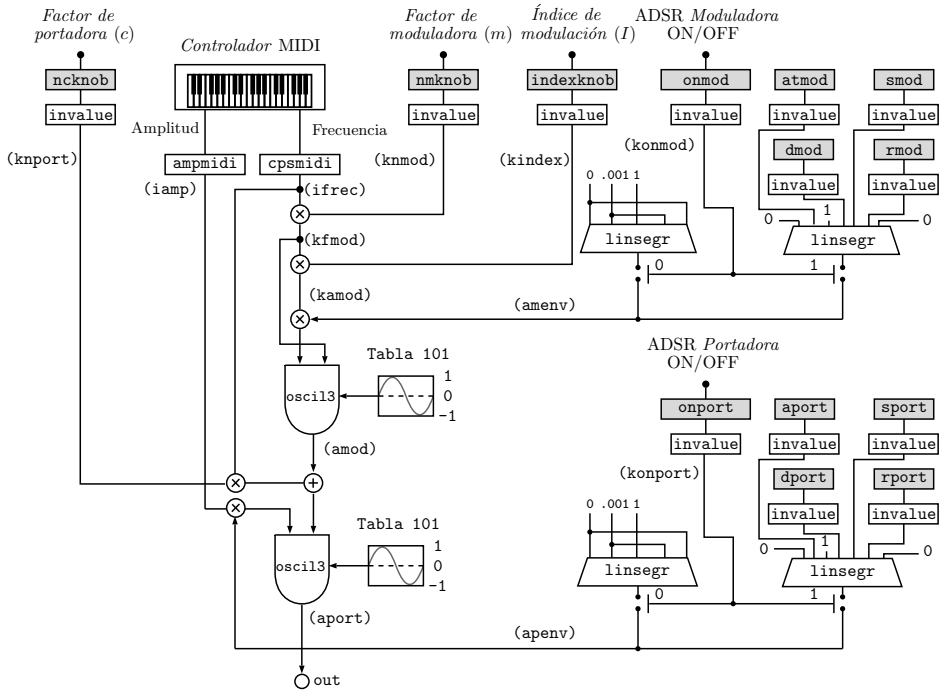


Figura 4.25: Diagrama de bloques detallado de la versión inicial del sintetizador FM (prototipo). Los bloques coloreados en gris hacen referencia a los *widgets*.

Una pequeña aproximación del sistema inicial podría ser la que se muestra en el diagrama de bloques simplificado (Fig. 4.24). Éste diagrama nos ofrece únicamente una visión global del sistema a través de sus entradas y salidas, por lo que no tendríamos en cuenta las funciones internas del sintetizador (sistema caja negra).

Además de las características básicas que presenta el prototipo, no debemos olvidar que debe estar controlado por un teclado vía MIDI, con lo que el sintetizador también recibirá la amplitud (o velocidad) y la frecuencia de la nota MIDI pulsada.

Para proceder a la implementación real del prototipo, será necesario seguir un diagrama algo más detallado en el que podamos estudiar su funcionamiento interno (sistema caja blanca). Por ello, se propone como solución el diagrama de la Fig. 4.25.

En primer lugar, el usuario introduce en tres *widgets* tipo *knob* denominados `indexknob`, `ncknob` y `nmknob`, los parámetros de entrada correspondientes al índice de modulación (I), factor de portadora (c) y factor de moduladora (m), respectivamente. A continuación, se leen los tres parámetros de los *widgets* mediante el *opcode* `invalue`, volcando los valores de cada parámetro en tres variables de tipo control (`k`): `kindex`, `knport` y `knmod`.

Otros dos *widgets* de tipo *checkbox*, cuyo nombre de canal será `onmod` y `onport`, recibirán un `1` o un `0` según el usuario desee activar o desactivar las envolventes ADSR para las señales moduladora y portadora, respectivamente. Como se trata de *widgets* de entrada de datos, debemos también leer sus valores a través de `invalue`, almacenándolos en dos variables `k`, una para cada señal: `konmod` y `konport`.

Por otra parte, debemos saber qué amplitud y frecuencia tendrá la nota MIDI que pulsemos en el controlador, por lo que emplearemos los *opcodes* `ampmidi` y `cpsmidi`.

El *opcode* `ampmidi` devuelve la amplitud o velocidad de la nota MIDI escalada al rango `0-iscal`, donde `iscal` es un factor de escalado que se introduce como único parámetro de entrada. Dado que nosotros deseamos que la amplitud varíe entre `0` y `1`, la variable `iscal` tomaría el valor `1`. El resultado de ésta amplitud escalada la almacenaríamos en una variable de tipo nota (`i`) a la que llamamos `iamp`. Dicha variable determinará la amplitud de la señal portadora.

El *opcode* `cpsmidi` devuelve la frecuencia (en Hz) de la nota MIDI

pulsada y no recibe ningún parámetro de entrada. Ésta frecuencia la almacenaríamos en la variable `ifrec`, que representará la frecuencia fundamental (f_0) de la nota.

A continuación, debemos calcular la amplitud y frecuencia que tendrá la señal moduladora. Para la frecuencia de la señal moduladora, tenemos que recordar la ecuación $f_0 = f_m/m$. Si despejamos el valor de f_m, la expresión quedaría $f_m = f_0 \cdot m$, donde f_0 = `ifrec` y m = `knmod`. Por lo tanto, podemos crear una nueva variable de tipo control (`kfmod`) que almacene la frecuencia de la moduladora: `kfmod = ifrec*knmod`.

Ahora tenemos que determinar la amplitud de la moduladora. Sabemos que el índice de modulación se puede calcular como $I = A_m/f_m$, y despejando la amplitud, la ecuación sería la siguiente: $A_m = I \cdot f_m$. Como el índice de modulación ya lo tenemos (`kindex`) y la frecuencia de modulación la acabamos de calcular (`kfmod`), podemos almacenar en otra variable de control (`kamod`) la amplitud de la moduladora: `kamod = kindex*kfmod`.

No obstante, la amplitud de la moduladora puede estar gobernada por una envolvente ADSR en caso de que el usuario lo haya decidido, es decir, si `konmod==1`. En ese caso, la amplitud de la moduladora `kamod` se vería multiplicada por dicha envolvente ADSR (que denotamos como `amenv`) antes de ser introducida en el oscilador.

Para llevar a cabo el cálculo de la envolvente, hemos recurrido a un *opcode* específico para uso real, denominado `linsegr`. La principal diferencia entre los *opcodes* de envolventes en tiempo real y los de envolventes controlados desde partitura, reside en que en los de tiempo real, la fase de relajación actúa cuando se recibe un mensaje MIDI de nota-*off*. Normalmente, los nombres de los *opcodes* de envolventes para tiempo real son iguales que los de control desde partitura, pero con una `r` al final del nombre. Por ejemplo, de `linseg`, `linsegr`[5].

El *opcode* `linsegr` permite construir una envolvente (tipo audio o control) formada por segmentos rectilíneos. Para especificar los segmentos, tenemos que indicar para cada uno de ellos su amplitud inicial, su duración (en segundos) y su amplitud final[6]. Por ello, siempre podemos crear una envolvente ADSR definiendo cuatro segmentos, correspondientes a cada una de las fases.

[5]Para más información sobre *opcodes* generadores de envolventes, consulte [9]

[6]Muy importante: `linsegr` recibe dichos parámetros como tipo nota (`i`). Por tanto, si tenemos que introducir una variable de tipo `k`, tendremos que hacer *casting*: `i(kmivariable)`.

Siguiendo el diagrama, podemos observar que si `konmod==1` (el usuario ha activado la envolvente de moduladora), se generará la envolvente ADSR especificada por el usuario a través de 4 *sliders* (`atmod`, `dmod`, `smod` y `rmod`). Las variables leídas de los *sliders* a través de `invalue` para cada una de las fases (que no aparecen en el diagrama por motivos de simplificación), serían `katmod` (ataque), `kdmod` (decaimiento), `ksmod` (amplitud de sostenimiento) y `krmod` (relajación).

La envolvente ADSR personalizada por el usuario, haciendo uso del `linsegr`, tendría la siguiente forma:

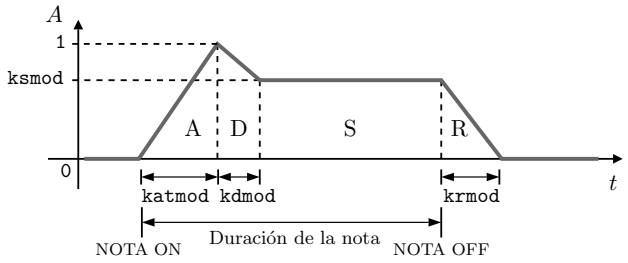


Figura 4.26: Envolvente ADSR especificada por el usuario haciendo uso del *opcode* `linsegr`.

Con lo cual, habríamos creado la envolvente de la señal moduladora: `amenv`.

Por otro lado, si el usuario no decide aplicar envolvente a la señal moduladora (`konmod==0`), lo que hacemos es simular una envolvente de amplitud constante igual a la unidad, aunque realmente será del tipo ASR, con tramos de ataque y relajación muy cortos (1 milisegundo). Evidentemente, con tiempos tan cortos para dichas fases, aparecerán *clicks* o chasquidos al pulsar y soltar la nota, con lo que se podrá estudiar el efecto que supone la presencia de una envolvente de amplitud.

Una vez determinada la envolvente de amplitud `amenv`, se multiplicará por `kamod`, obteniendo finalmente la amplitud definitiva de la señal moduladora.

Si continuamos el diagrama, podemos reconocer el *opcode* `oscil3`, que generará las señales portadora y moduladora. Éste *opcode* permite sintetizar la señal empleando interpolación cúbica, por lo que la señal de salida tendrá mayor calidad que la que tendría empleando otros *opcodes* como `oscil`[7]. Toma como parámetros de entrada la amplitud, frecuencia y tabla de onda, devolviendo a su salida una señal de tipo `a` o `k`.

[7]En [10] encontrará un breve estudio comparativo realizado para los osciladores que ofrece *Csound*.

En el caso que nos ocupa, el primer `oscil3` (modulador) tomará como amplitud `kamod*amenv`, como frecuencia `kfmod` y como tabla, la `101` de tipo sinusoidal. La señal moduladora generada, se almacenará en la variable de tipo audio `amod`.

En éste momento, necesitaríamos obtener la amplitud y frecuencia de la señal portadora. Como se trata de una modulación en frecuencia simple, la señal moduladora (`amod`) deberá sumarse a la frecuencia de la portadora.

La frecuencia de la portadora se puede calcular siguiendo el mismo planteamiento que con la moduladora: como $f_0 = f_c/c$, entonces $f_c = f_0 \cdot c$. Ésto quiere decir que la frecuencia de la portadora será `ifrec*knport`, donde `knport` se obtiene del *widget* `ncknob`. Por consiguiente, la frecuencia definitiva de la portadora (incluyendo modulación) será `ifrec*knport+amod`.

En lo que respecta a la amplitud de la portadora, ésta se podrá calcular como `ifrec*apenv`, donde `apenv` hace referencia a la envolvente de la portadora que se determina del mismo modo que la envolvente de la portadora.

Introducimos, finalmente, los parámetros de amplitud y frecuencia en el segundo oscilador, así como la misma tabla de onda sinusoidal que en el modulador, obteniendo la señal de salida del sintetizador. Solo quedaría llevarla a la salida de la tarjeta de audio mediante el *opcode* `out` (por ser monofónica en ésta versión inicial).

4.2.2. Interfaz gráfica del prototipo

Aunque en el diagrama de bloques se ha propuesto una serie de *widgets* para introducir los parámetros de entrada del sintetizador, no hemos visto su aspecto real. Para conformar la interfaz gráfica del sistema inicial, nos hemos decantado por un formato tipo *rack* (rejilla), de modo que se pueden ir añadiendo nuevos módulos que cumplan determinadas funciones de una forma cómoda (Fig. 4.27).

Figura 4.27: Interfaz gráfica de la versión inicial del sintetizador FM.

En la interfaz podemos distinguir dos módulos, separados por una línea horizontal. El módulo superior se corresponde con el control del índice de modulación (I) y los factores de portadora (c) y moduladora (m). El módulo inferior hace referencia a los generadores de envolventes para las señales portadora y moduladora.

Para el módulo superior, se han utilizado 3 *widgets* tipo *knob* a los que se han enlazado otros 3 *widgets display*, para poder visualizar el valor que el usuario desea introducir. Para producir el enlace, debemos darle el mismo nombre al *display* que al *knob* al que lo queremos enlazar. Por tanto, los *display* tendrán como nombre de canal: `indexknob`, `ncknob` y `nmknob`.

Respecto a los valores límite de los tres potenciómetros, hemos escogido los siguientes:

- `indexknob` (I): Desde `0` a `20`.
- `ncknob` (c): Desde `0` a `10`.
- `nmknob` (m): Desde `0` a `10`.

El motivo de ésta elección reside en que necesitamos un mayor rango para el índice de modulación, ya que por una parte, es más fácil estudiar su efecto con un rango amplio; y por otra, facilitará el diseño sonoro, puesto que determinados instrumentos precisarán de un índice alto, así como otros de un índice bajo o intermedio.

En cambio, los factores de frecuencia no requieren llegar a un límite tan elevado, ya que éstos, en definitiva, multiplicarán a la frecuencia fundamental de la nota que estemos pulsando en el controlador MIDI (`ifrec`). Por consiguiente, si pulsamos una tecla cuya frecuencia es elevada y todavía la multiplicamos por un factor alto, es posible que la frecuencia resultante se salga del rango auditivo (> 20 kHz), lo cual no es favorable.

Cabe destacar que con los límites escogidos para los factores de frecuencia se pueden conseguir numerosas combinaciones de relaciones $c : m$, con lo que no limitaremos el diseño de los instrumentos.

Pasando al módulo inferior, podemos apreciar que se han utilizado para cada generador: 4 *sliders*, 4 *displays* y un *checkbox*.

Cada uno de los 4 *sliders* hacen referencia (como ya vimos en el apartado anterior) a las cuatro fases de una envolvente tipo ADSR. Los límites elegidos en éste caso para los deslizadores o *faders* han sido los siguientes:

Portadora:

- `aport` (A): Desde `0.005` hasta `1`.
- `dport` (D): Desde `0.001` hasta `1`.
- `sport` (S): Desde `0` hasta `1`.
- `rport` (R): Desde `0.005` hasta `1`.

Moduladora:

- `atmod` (A): Desde `0.005` hasta `1`.
- `dmod` (D): Desde `0.001` hasta `1`.
- `smod` (S): Desde `0` hasta `1`.
- `rmod` (R): Desde `0.005` hasta `1`.

En primer lugar, podemos observar que los valores límite para los *widgets* del generador de envolvente de portadora son idénticos a los de la moduladora.

Con respecto a la fase de ataque, hemos optado por un límite inferior de 5 ms y uno superior de 1 s. El hecho de haber seleccionado 5 ms y no 0 s se debe a que en éstos *opcodes*, si fijamos un valor nulo para la fase de ataque, la envolvente de amplitud será nula, por lo que la señal no se escuchará. Además, hemos podido comprobar de manera experimental que 5 ms es el tiempo a partir del cual no se producen *clicks* al activar la nota desde el controlador.

Por otra parte, el límite superior de 1 segundo es suficiente como para obtener un ataque muy lento, aunque en caso de que así se decida, podría ampliarse.

La fase de decaimiento la hemos establecido entre 1 ms y 1 s. En ésta ocasión, el límite inferior de 1 ms nos permitirá diseñar instrumentos que tengan un ataque rápido (aunque podrían aparecer *clicks*), como por ejemplo, sonidos percusivos.

La amplitud de sostenimiento debe variar entre 0 y 1, de modo que si dicha amplitud es nula, no se escuchará señal. Si la amplitud de sostenimiento es igual a la unidad, la fase de decaimiento no tendrá ningún efecto en la señal.

Finalmente, la fase de relajación se ha diseñado de igual modo que la fase de ataque.

4.2.3. Código fuente

Por tratarse únicamente de la versión inicial del sistema, hemos creído conveniente mostrar el código fuente del sintetizador. No obstante, cuando estudiemos la versión definitiva del sistema, el código fuente asociado figurará en el apéndice A.

El código propuesto (cuya implementación ha sido estudiada en apartados anteriores) sería el que se muestra a continuación:

```
;Diseño del sintetizador FM simple (Prototipo).
<CsoundSynthesizer>
;Diseño de la orquesta.
<CsInstruments>
    ;Declaramos las variables globales.
    sr = 44100 ;Frecuencia de muestreo.
    ksmps = 128 ;Número de muestras por periodo de control.
    nchnls = 1 ;Monofónico.
    ;Sintetizador FM a partir de dos osciladores de interpolación cúbica.
    instr 1
        ;Amplitud de la nota MIDI normalizada.
        iamp ampmidi 1
        ;Frecuencia de la nota MIDI.
        ifrec cpsmidi
        ;Leemos el estado de los interruptores ON de las envolventes.
        konmod invalue "onmod"
        konport invalue "onport"

        ;Almacenamos los parámetros de índice de modulación y factores de
        ;frecuencia de portadora y moduladora.
        kindex invalue "indexknob"
        knmod invalue "nmknob"
        knport invalue "ncknob"

        ;Calculamos la frecuencia y amplitud de la señal moduladora.
        kfmod = ifrec*knmod
        kamod = kindex*kfmod

        ;Envolvente de la señal moduladora.
        if (konmod==1) then ;Si se activa la casilla ON.
            ;Leemos los parámetros ADSR de los faders.
            katmod invalue "atmod" ;Tiempo de ataque (s).
            kdmod invalue "dmod" ;Tiempo de decaimiento (s).
            ksmod invalue "smod" ;Amplitud de sostenimiento.
            krmod invalue "rmod" ;Tiempo de relajación (s).

            ;Generamos la envolvente correspondiente de la señal moduladora.
            amenv linsegr 0, i(katmod), 1, i(kdmod), i(ksmod), i(krmod), 0

        else ;Si se desactiva la casilla ON.
            amenv linsegr 0,0.001,1,0.001,0 ;Envolvente de amplitud unidad.
        endif
```

```
        ;Envolvente de la señal portadora.
        if (konport==1) then ;Casilla ON activada.

            ;Obtenemos los parámetros ADSR de los faders.
            kaport invalue "aport" ;Tiempo de ataque (s).
            kdport invalue "dport" ;Tiempo de decaimiento (s).
            ksport invalue "sport" ;Amplitud de sostenimiento.
            krport invalue "rport" ;Tiempo de relajación (s).

            ;Generamos la envolvente correspondiente de la señal portadora.
            apenv linsegr 0, i(kaport), 1, i(kdport), i(ksport), i(krport), 0

        else ;Casilla ON desactivada.
            apenv linsegr 0,0.001,1,0.001,0 ;Envolvente de amplitud unidad.
        endif

        ;Señal moduladora.
        amod oscil3 kamod*amenv, kfmod, 101
        ;Señal portadora sin LFO aplicado.
        aport oscil3 iamp*apenv, ifrec*knport+amod, 101
        ;Mandamos la señal a la salida de la tarjeta de audio.
        out aport
    endin
</CsInstruments>

;Diseño de la partitura.
<CsScore>
    ;Definimos las tablas necesarias.
    f 1 0 4096 10 1 ;Tabla de onda sinusoidal (GEN 10).
    ;Llamamos al instrumento creado.
    i 1 0 3600 ;1 hora de ejecución.
    e
</CsScore>
</CsoundSynthesizer>
```

4.2.4. Experimentación y validación

Hasta ahora, disponemos de una versión inicial de un sintetizador FM simple. Para comprobar que la implementación es correcta, se someterá al sistema a diversas pruebas específicas, verificando que todos los módulos responden de forma adecuada.

4.2.4.1. Control del índice de modulación

La primera prueba que realizaremos consistirá en ejecutar varios acordes en el controlador MIDI y, fijados los factores de portadora y moduladora en $c = 1$ y $m = 0,5$, iremos modificando progresivamente el índice de modulación I, partiendo desde el valor nulo ($I = 0$, sin modulación), hasta alcanzar el valor máximo ($I = 20$) (Fig. 4.28). Por último y tras haber alcanzado el valor máximo del índice, realizaremos el proceso inverso, por lo que pasaremos de un sonido rico en armónicos a una sinusoidal simple.

Éste procedimiento lo realizaremos varias veces para apreciar con más detalle el efecto producido. También debemos destacar que aunque activaremos los generadores de envolventes, solo se emplearán para simular una envolvente unidad, de modo que el sonido quede libre de chasquidos al principio y final de la pulsación de la nota.

Figura 4.28: Ajuste inicial de parámetros antes de verificar el control del índice de modulación.

En el siguiente fichero de audio, se puede escuchar la prueba realizada:

15.-Índice de modulación.mp3

El funcionamiento es el adecuado puesto que se produce la generación de parciales a medida que aumentamos el valor del índice de modulación. En cambio, al reducir el valor del índice de modulación, se observa cómo van eliminándose componentes frecuenciales de la señal, hasta llegar a la frecuencia de la señal portadora cuando el índice es nulo.

Por último, en la Fig. 4.29 se puede observar un tramo de la señal generada (entre el segundo 6 y $6,25$), en el que comienza a modularse en FM debido al aumento del índice de modulación.

4.2.4.2. Variación del factor de portadora

En ésta segunda prueba, verificaremos el comportamiento de la señal cuando variamos el factor de portadora. Por tanto, será necesario tener índice de modulación nulo para que no exista modulación alguna y, aunque no sea necesario, también pondremos a cero el factor de moduladora. El ajuste de las envolventes será idéntico con respecto a la prueba anterior (Fig. 4.30).

Se pulsará una única nota y, seguidamente, se modificará progresivamente el factor de portadora, desde el valor $c = 0$ hasta $c = 10$. Por último, se retornará de nuevo al valor $c = 0$, sin detener en ningún momento la pulsación de la nota durante el transcurso de la prueba.

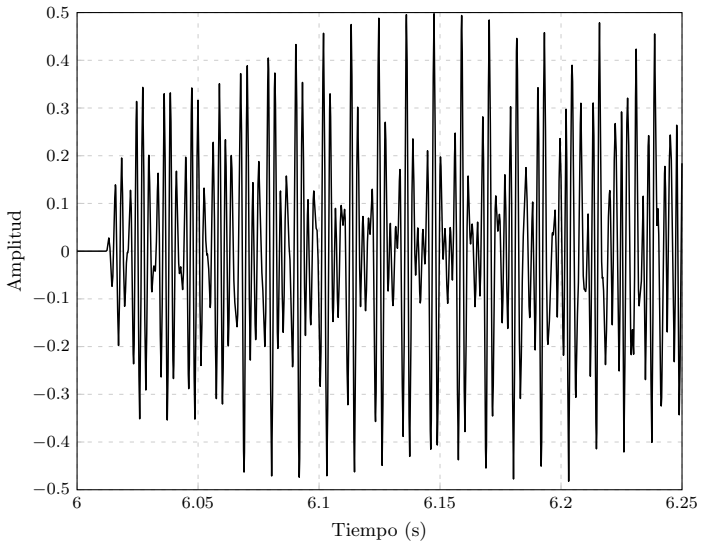


Figura 4.29: Prueba del control del índice de modulación. La señal comienza a modularse en frecuencia de forma progresiva, añadiendo nuevos armónicos.

Figura 4.30: Ajuste inicial de parámetros antes de verificar la variación del factor de portadora.

Acceda al siguiente archivo de audio para comprobar el efecto de éste parámetro:

16.-Factor de portadora.mp3

Dado que el factor de portadora multiplica la frecuencia fundamental de la nota que estamos pulsando, podemos percibir que la frecuencia de la señal de salida aumenta según elevamos el factor de portadora (Fig. 4.31). Por otro lado, cuando reducimos el factor de portadora, también disminuirá la frecuencia de la señal resultante (Fig. 4.32).

También debemos saber que si $c = 0$, no se escuchará ninguna señal debido a que multiplicaríamos por cero la frecuencia fundamental.

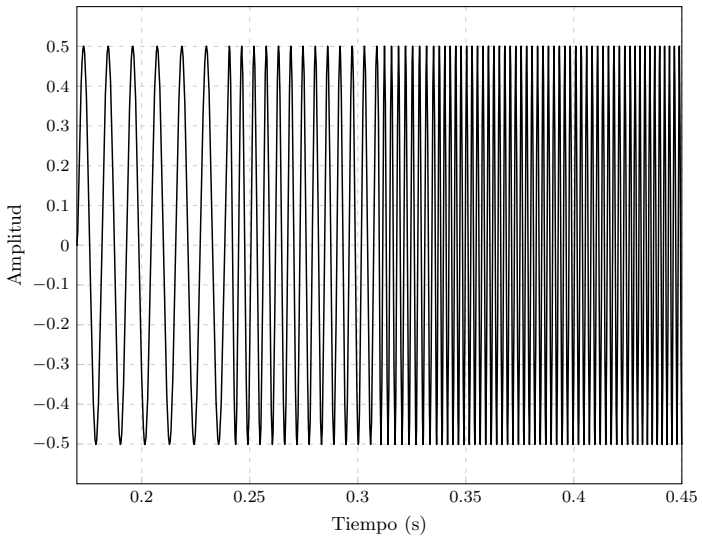


Figura 4.31: Prueba del control del factor de portadora (1). La señal de salida aumenta su frecuencia al elevar el factor de portadora.

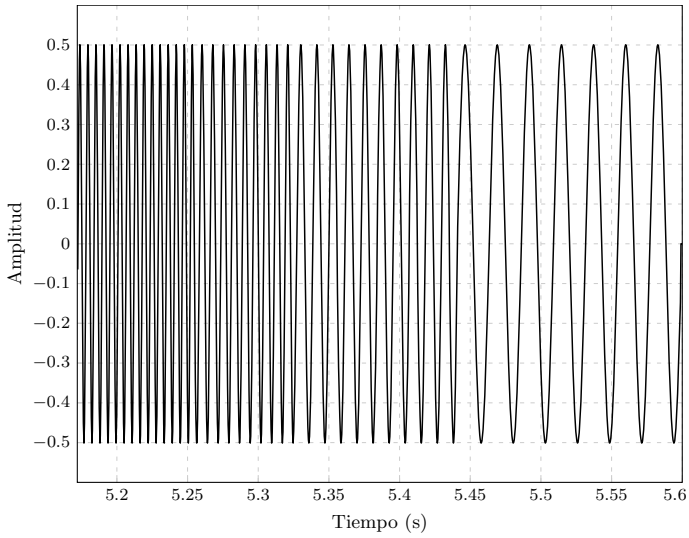


Figura 4.32: Prueba del control del factor de portadora (2). La señal de salida disminuye su frecuencia al reducir el factor de portadora.

4.2.4.3. Variación del factor de moduladora

Para comprobar el funcionamiento del módulo referente al factor de moduladora (m), será indispensable llevar a cabo una modulación, ya que si el índice de modulación es nulo ($I = A_m/f_m = 0$), la amplitud de la moduladora será nula ($A_m = 0$). Por tanto, el hecho de variar m no producirá ningún efecto, ya que no existirá señal moduladora.

Para llevar a cabo la prueba, fijamos el valor del índice de modulación y el factor de portadora a la unidad ($I = c = 1$), pulsaremos una nota del teclado y desplazaremos progresivamente el potenciómetro del factor de moduladora desde $m = 0$ hasta $m = 10$, regresando otra vez a $m = 0$ (véase Fig. 4.33).

Figura 4.33: Ajuste inicial de parámetros antes de verificar la variación del factor de moduladora.

En el fichero de audio que se muestra a continuación, puede oírse la prueba ejecutada:

17.-Factor de moduladora.mp3

Si abrimos el fichero, podremos apreciar que dentro de la señal de salida se perciben dos señales. La primera señal es la portadora y la reconoceremos porque su frecuencia es constante e igual a la de la nota pulsada. La segunda sería la señal moduladora, que irá subiendo y bajando su frecuencia a medida que aumentamos o reducimos el factor de moduladora. Éste hecho nos indica que funciona correctamente el módulo.

En la Fig. 4.34, se muestra un pequeño instante de tiempo en el que la señal de salida cambia su forma de onda por la modificación del factor de moduladora.

4.2.4.4. Generador de envolvente de portadora

A continuación, analizaremos el generador de envolvente de la señal portadora. En éste caso, no realizamos modulación en frecuencia, de modo que tanto I como m serán nulos y asignaremos como factor de portadora el valor 1.

A continuación, activamos el generador de envolvente de portadora y creamos una envolvente ADSR con los siguientes parámetros: A = 0,337 s, D = 0,625 s, S = 0,479 y R = 0,565 s (Fig. 4.35).

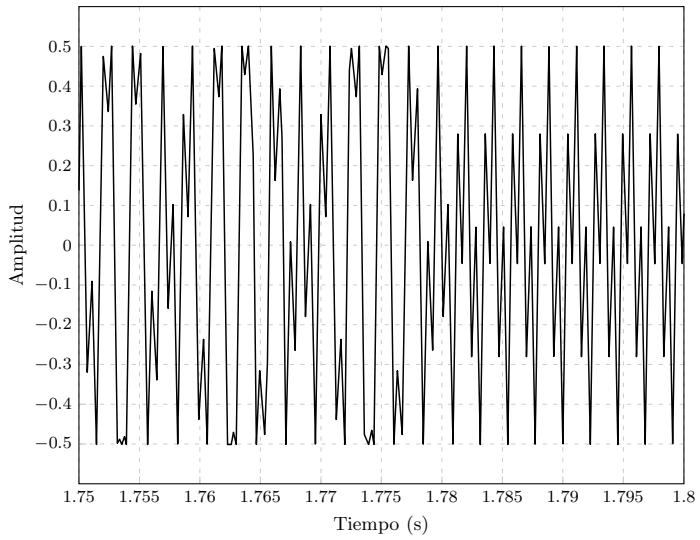


Figura 4.34: Cambio en la forma de onda de la señal resultante por la modificación del factor de moduladora (m).

Figura 4.35: Ajuste inicial de parámetros antes de comprobar el generador de envolvente de portadora (configuración 1).

Escuchemos ahora el resultado en el siguiente archivo de audio:

18.-Envolvente de portadora 1.mp3

El sonido comienza aumentando su amplitud hasta un punto máximo (fase de ataque), pero a partir de éste momento, desciende ligeramente durante poco más de medio segundo (fase de decaimiento) hasta situarse en la amplitud de sostenimiento. Por último, podemos observar que la señal decae en amplitud hasta ser imperceptible (fase de relajación).

Para contrastar mejor las distintas fases de la envolvente, hemos representado diversas partes de la señal. Como vemos, los parámetros escogidos para la envolvente ADSR se corresponden con la señal obtenida (ver Fig. 4.36, 4.37 y 4.38).

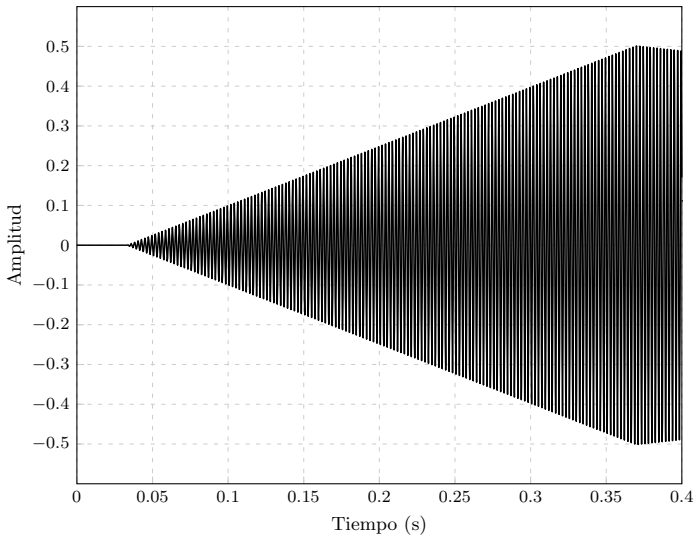


Figura 4.36: Señal de salida con envolvente ADSR aplicada. Tramo correspondiente a la fase de ataque.

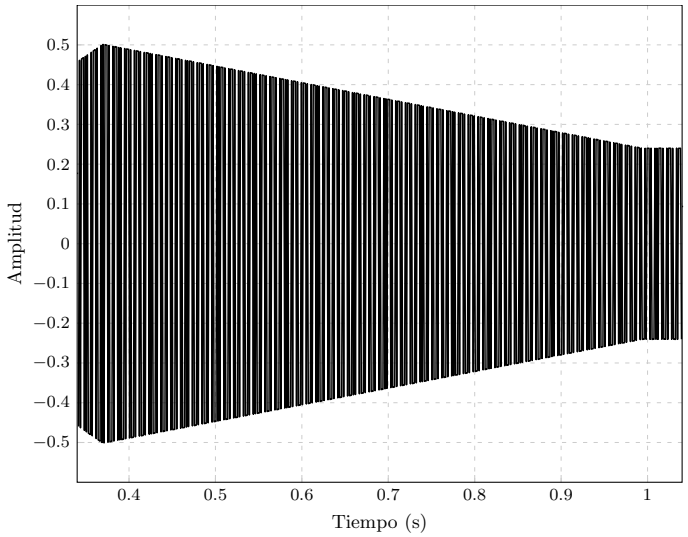


Figura 4.37: Señal de salida con envolvente ADSR aplicada. Tramo correspondiente a la fase de decaimiento (y parte de sostenimiento).

Para finalizar la comprobación del generador de envolvente, diseñaremos otra envolvente de amplitud, pero ésta vez con un ataque más largo (A = 1 s) y tiempos mínimos para las fases de decaimiento y relajación (D = 0,001 s y R = 0,005 s). La amplitud de sostenimiento se situará en el valor máximo posible (S = 1) (Fig. 4.39).

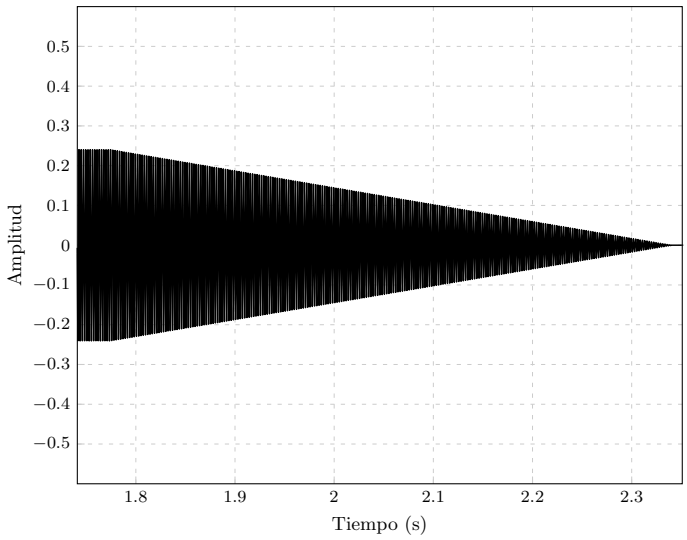


Figura 4.38: Señal de salida con envolvente ADSR aplicada. Tramo correspondiente a la fase de relajación (y parte de sostenimiento).

Figura 4.39: Ajuste inicial de parámetros antes de comprobar el generador de envolvente de portadora (configuración 2).

En éste caso, la señal de salida aumenta su amplitud lentamente hasta alcanzar la amplitud de sostenimiento. Al estar situada dicha amplitud a su valor máximo, no habría diferencia entre la amplitud de pico (1) y la de sostenimiento, con lo que el parámetro de decaimiento no tendría efecto. Finalmente, el sonido reduce su amplitud en un intervalo muy corto de tiempo debido al escaso valor de R.

Oiga el ejemplo en el fichero de audio que se muestra a continuación:

19.-Envolvente de portadora 2.mp3

Con los dos ejemplos incluidos en éste apartado, podemos deducir que el generador de envolvente de la señal portadora responde según lo esperado.

4.2.4.5. Generador de envolvente de moduladora

Para poder verificar que el funcionamiento del generador de envolvente de moduladora es correcto, siempre tendremos que realizar una modulación en frecuencia, es decir, no pueden ser nulos ni el índice de modulación ni el factor de moduladora.

Ésto se debe a que si $I = 0$ o $m = 0$, no existirá moduladora, con lo que la envolvente de moduladora no tendrá efecto. Dicha envolvente permitirá introducir dinámica en la modulación.

Empecemos con una pequeña prueba, observando qué sucede cuando establecemos los siguientes parámetros en el módulo superior: $I = 5,8$, $c = 1$ y $m = 2$. El generador de envolvente de portadora lo emplearemos para generar una envolvente constante (igual a la unidad), mientras que el generador de envolvente de moduladora lo utilizaremos para crear una envolvente ADSR con los siguientes valores: A = 0,627 s, D = 0,563 s, S = 0,396 y R = 0,005 s (Fig. 4.40).

Figura 4.40: Ajuste inicial de parámetros antes de comprobar el generador de envolvente de moduladora (configuración 1).

Detengámonos en éste punto para escuchar la prueba llevada a cabo:

20.-Envolvente de moduladora 1.mp3

Primeramente, podemos destacar que el sonido inicial es producto de una modulación FM, puesto que $I > 0$ y $m > 0$. Seguidamente, observamos que la señal va ganando parciales progresivamente, durante la fase de ataque de la envolvente de la señal moduladora.

Éste fenómeno se debe a que al aumentar la amplitud de la señal moduladora mientras está fijada la frecuencia de modulación, también se incrementa el índice de modulación, consiguiendo un sonido más rico en armónicos.

En la fase de decaimiento de la envolvente de moduladora, se puede apreciar cómo el sonido reduce su número de parciales hasta llegar a estabilizarse, por el decremento de amplitud de la moduladora hasta alcanzar la fase de sostenimiento.

El sonido finaliza de forma brusca al soltar la nota debido a que el tiempo de relajación de ambas envolventes es muy pequeño (5 ms).

A continuación, se ilustra una pequeña porción del comienzo de la señal. Durante éste instante de tiempo, la señal de salida incrementa el número de armónicos debido a la fase de ataque de la envolvente de modulación:

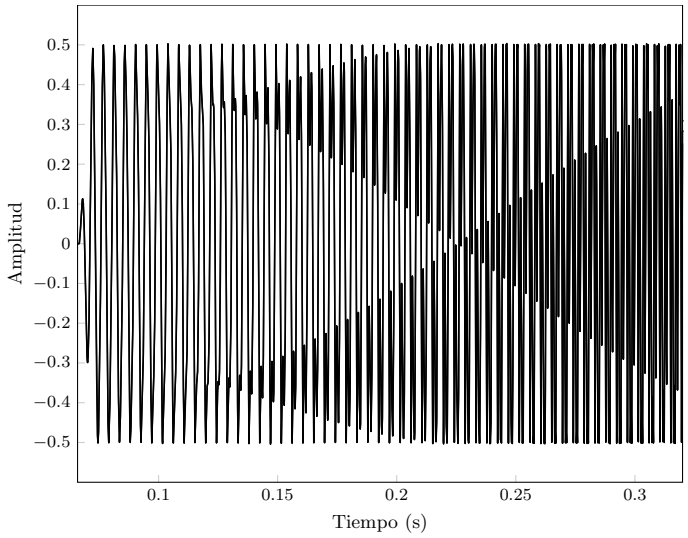


Figura 4.41: Incremento de parciales de la señal de salida durante la fase de ataque de la envolvente de modulación.

La segunda y última prueba que realizaremos consistirá en evaluar la señal de salida cuando programamos una envolvente ADSR para la señal moduladora con los parámetros que siguen: A = 0,005 s, D = 1 s, S = 0,354 y R = 0,295 s. Además, se ha ajustado el tiempo de relajación de la envolvente de la portadora a R = 0,544 s, a fin de no cortar la fase de relajación a la envolvente moduladora (ver Fig. 4.42).

El resultado de ésta prueba puede escucharse en el fichero adjunto:

`21.-Envolvente de moduladora 2.mp3`

A diferencia de la prueba anterior, en éste caso el sonido comienza con un

Figura 4.42: Ajuste inicial de parámetros antes de comprobar el generador de envolvente de moduladora (configuración 2).

número determinado de parciales, y va reduciéndose durante 1 segundo hasta alcanzar la fase de sostenimiento. Llegados a éste punto, donde la modulación es constante, se mantiene aproximadamente otro segundo hasta la fase de relajación, donde en casi 300 ms la señal pierde todavía más armónicos de forma gradual, hasta que la señal de salida se extingue.

Al haber comprobado que la versión inicial del sintetizador FM simple responde a las exigencias planteadas, ya estamos en condiciones de dotar al sistema con nuevas características.

4.3. Versión definitiva del sintetizador

Un sintetizador FM simple ofrece una ligera idea del espectro que podemos llegar a conseguir utilizando únicamente dos osciladores sinusoidales. Incluyendo, además, una envolvente de amplitud para cada uno de ellos, permitiremos que la señal resultante no solo gane dinámica, sino una modulación más compleja.

Por otro lado, si deseamos que nuestro sintetizador sea más versátil y, por tanto, que ofrezca mayor capacidad de diseño sonoro, será requisito indispensable añadirle nuevas opciones.

Nuestro prototipo solo posibilita el uso de tablas de onda de tipo sinusoidal. Sería interesante estudiar su comportamiento si pudiéramos elegir qué forma de onda deseamos que genere tanto la señal portadora como la moduladora.

Asimismo, un módulo que sea capaz de analizar el espectro de la señal de salida en tiempo real, facilitaría en gran medida un aprendizaje gráfico de la técnica de síntesis FM, observando la influencia de la variación de los distintos parámetros (I, c, m,...etc). En éste sentido, ésto constituiría un apoyo a la labor docente.

Para mejorar la experiencia, podríamos incluir un módulo de efectos digitales que incorpore un oscilador de baja frecuencia (LFO), retardo (*delay*) y reverberación (*reverb*). Cada submódulo tendría que permitir la modificación de los parámetros típicos, así como la posibilidad de encendido y apagado.

Un potenciómetro regulable que ajuste el volumen final de la señal de salida (MASTER) siempre puede ser interesante. Así pues, debería ir acompañado de un LED que informe al usuario sobre si la señal distorsiona (CLIP). Ésta capacidad es muy frecuente en las mesas de mezclas.

Para comprobar que la conexión entre nuestro controlador MIDI y *QuteCsound* se lleva a cabo de forma satisfactoria, un sensor que indique si se recibe una nota-ON MIDI o no podría ser de ayuda para la detección de problemas.

Finalmente, se podría implementar un pequeño selector de *presets*, que permita cargar las configuraciones de parámetros que hayamos almacenado con anterioridad.

Por consiguiente, las nuevas características añadidas al sistema para crear la versión definitiva del sintetizador FM serán las siguientes:

- Selector de forma de onda para portadora y moduladora: sinusoidal, cuadrada, triangular, diente de sierra y pulso.
- Analizador de espectro en tiempo real de la señal de salida, con posibilidad de encendido o apagado.
- LFO que permita modular amplitud (*trémolo*) o frecuencia (*vibrato*), con amplitud y frecuencia variables, así como opción para ON/OFF.
- Módulo de *delay* con los parámetros ajustables: tiempo de retardo (ms), realimentación o *feedback* y *dry/wet*. Se podrá encender o apagar (*bypass*) según la decisión del usuario.
- Módulo de *reverb* con factor de tamaño de la sala ajustable, así como atenuación en alta frecuencia variable y *dry/wet*. La opción de ON/OFF también se deberá implementar.
- Potenciómetro de ajuste de ganancia de la señal de salida (MASTER) con LED que se iluminará en rojo en caso de que la señal distorsione.
- Selector de *presets*.
- LED de nota-ON MIDI (naranja), que se encenderá en caso de que se reciba un mensaje de nota-ON y se apagará en caso contrario.

- Botones de encendido y apagado del sintetizador, para comenzar la ejecución del programa y detenerla, respectivamente.
- LED ON (verde) que informa si el sintetizador está encendido (el programa se está ejecutando).

4.3.1. Diagrama de bloques

El diagrama de bloques simplificado que hemos propuesto para incluir las nuevas capacidades del sistema es el que se muestra en la figura adjunta:

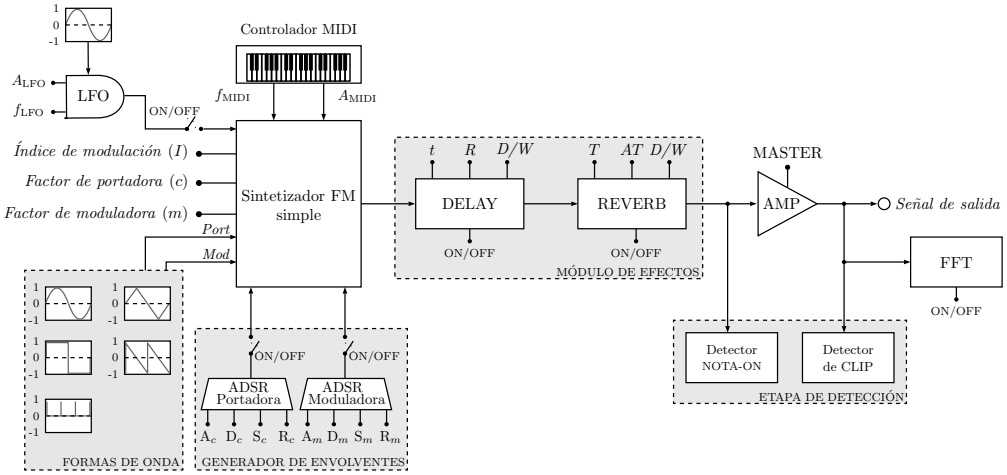


Figura 4.43: Diagrama de bloques simplificado de la versión definitiva del sintetizador FM.

Se trata de una ampliación del diagrama referente al prototipo (Fig. 4.24), en el que podemos distinguir diferentes etapas o módulos, anteriores a la entrada y posteriores a la salida del sintetizador.

Primeramente, se introducen todos los parámetros que ya incluíamos en el prototipo: índice de modulación (I), factores de portadora y moduladora (c y m), envolventes ADSR para ambas señales con sus respectivos valores y amplitud y frecuencia de la nota MIDI pulsada desde el controlador.

Como vemos, el usuario puede llevar a cabo una modulación en baja frecuencia sobre la señal de la portadora, en caso de que active el LFO. Para ello, será necesario que introduzca los valores de la amplitud y frecuencia del LFO. Finalmente, tendrá que seleccionar (aunque no aparezca en el diagrama) si el LFO actuará como modulador de amplitud sobre la portadora (provocando un *trémolo*), o bien si se comportará como modulador en baja frecuencia (realizando un *vibrato*).

Más abajo se puede distinguir un selector de formas de onda, tanto para la señal portadora como para la moduladora. El sintetizador debe saber qué onda debe generar con cada oscilador, por lo que el usuario deberá decidir alguna de las cinco posibles. En cualquier caso, se pueden seleccionar de forma independiente para la portadora y moduladora.

Cuando ya tenemos calculada la señal de salida del sintetizador FM simple, nos encontramos con la etapa de efectos digitales, donde vemos una unión en cascada de un módulo DELAY con otro de REVERB. Para ambos dispositivos tenemos la opción de activarlos o desactivarlos (ON/OFF), de modo que se pueda aplicar dos, uno o ningún efecto. Realmente, el modo OFF de dichos módulos realizan la función de *bypass*, es decir, no aplican efecto a la señal de entrada, con lo que sale del dispositivo inalterada.

El bloque correspondiente al DELAY (o retardo) permite aplicar un tiempo de retardo t (en ms) sobre la señal de entrada. La realimentación (o *feedback*, R) hace referencia a cuánto tardará en disiparse la señal retardada. Nosotros lo hemos medido en porcentaje (%), aunque también es habitual verlo expresado en tanto por 1.

Cuando la realimentación sea 0 % (o 0), la señal retardada se esfumará rápidamente, con lo que en la práctica escucharemos que la señal retardada solo suena una vez. En cambio, si la realimentación toma el valor 100 % (o 1), la señal retardada se repetiría infinitas veces.

Como último ajuste para el DELAY, tendríamos el *dry/wet* (D/W), cuya función es controlar la mezcla de señal seca (sin efecto aplicado) y señal retardada. También se expresa en porcentaje, con lo que, para 0 % tendríamos señal completamente seca, mientras que para 100 % obtendríamos solo señal retardada.

Para completar la etapa de efectos, nos topamos con el módulo de reverberación. La reverberación es un efecto que se produce por las reflexiones del sonido directo en una sala debido a los obstáculos (como por ejemplo las paredes). El receptor recibe una suma del sonido directo y las reflexiones producidas, de modo que no se perciben dos sonidos distintos como en el *delay*, sino un único sonido.

El primer parámetro del REVERB que podemos modificar es el factor del tamaño de sala (T) (entre 0 y 1). Cuanto más grande sea T, simularemos una sala con mayor volumen, con lo que el tiempo de reverberación aumentará, así como la persistencia sonora de las reflexiones.

El segundo parámetro es el factor de atenuación de altas frecuencias (AT, entre 0 y 1) y estaría relacionado con el decaimiento de la amplitud según el rango de frecuencias. Por ello, un valor de $AT = 0$ significaría que todas las frecuencias decaen a la misma velocidad, mientras que si aumentamos la magnitud de AT, las frecuencias más elevadas se atenuarán antes que las medias y bajas.

Finalmente, tenemos de nuevo el control de *dry/wet*, que nos permitirá manejar la mezcla entre señal seca y reverberada.

El detector de nota-ON MIDI tiene que comprobar si se ha recibido un mensaje MIDI de nota-ON (144), y en ese caso, deberá iluminar un LED indicador. Si no se recibiera dicho mensaje, el LED se mantendría apagado.

A continuación, vemos un amplificador (AMP), que permitirá aplicar un factor de ganancia (MASTER) a la señal de salida, tras haber pasado el módulo de efectos.

Para finalizar la etapa de detección, tenemos el detector de CLIP o distorsión. Éste dispositivo tiene que analizar la señal de salida amplificada en pequeños intervalos de tiempo, de modo que obtenga el valor máximo o mínimo de la señal (en valor absoluto) y en caso de que dicho valor supere o iguale la unidad, se avisará mediante un LED encendido, indicando que la señal está distorsionada. En caso de que no supere la unidad, el LED se mantendrá apagado.

Antes de enviar la señal resultante a la salida, existirá la posibilidad de llevar a cabo el análisis de su espectro mediante un módulo que calcule su *Transformada Rápida de Fourier* (*FFT, Fast Fourier Transform*) y permita su representación en tiempo real.

Cabe destacar que el selector de *presets* no se ha incluido en el diagrama de bloques, ya que es se trata de una función externa al código, puesto que solo afecta al panel de *widgets*. Los botones de encendido y apagado, así como el *display* referente al estado ON del sintetizador se estudiarán en el próximo apartado.

4.3.2. Implementación de las nuevas funciones

En la presente sección, se procederá a la implementación de las distintas etapas vistas en el diagrama simplificado.

Para empezar, y enfocándonos en el código de *QuteCsound*, hemos decidido separar la implementación de la orquesta, modularizándola en 4

instrumentos:

- `instr 1`: Es el primer instrumento y se trata de una modesta ampliación del instrumento `instr 1` del prototipo. Englobará los osciladores de portadora y moduladora, los generadores de envolventes, el LFO y los selectores de forma de onda.
- `instr 100`: Se trata del segundo instrumento de la orquesta y constituye el módulo de DELAY.
- `instr 101`: Es el tercer instrumento que implementaremos y se caracteriza por contener el dipositivo de REVERB.
- `instr 102`: Será el último instrumento y almacenará la etapa de detección, amplificación y representación del espectro en tiempo real.

La mecánica que se seguirá para la generación del sonido será crear una variable global de audio (`gabus`) que se inicializará a cero en la cabecera de la orquesta, de modo que cada instrumento vaya "incorporándole" a dicha variable los cálculos que correspondan.

Ésta variable constituye el *bus* o canal por el que se transmitirá el sonido a lo largo de todos los instrumentos de la orquesta, hasta llegar a la salida de la tarjeta de audio.

Cuando la variable se transmita a la tarjeta de audio y podamos oír la señal sintetizada, el instrumento `102` volverá a inicializar el bus global de audio a cero (`gabus=0`), antes de volver a repetir el proceso (ver Fig. 4.44).

Si no finalizamos la variable estableciéndola a un valor nulo, se produciría realimentación positiva, de modo que el sonido generado iría aumentando su amplitud en cada iteración (`gabus` se autoincrementaría), por almacenar continuamente los valores del bus de todos los procesos anteriores.

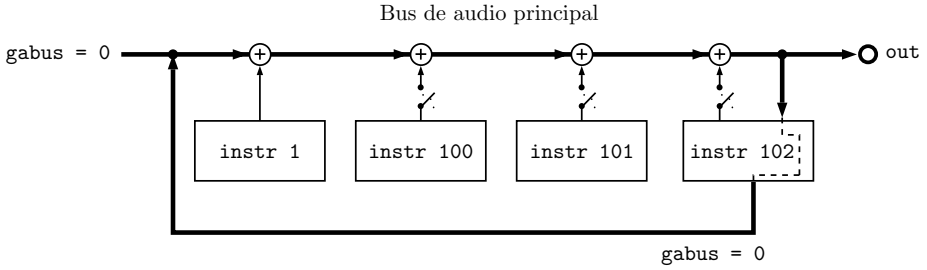


Figura 4.44: Proceso básico para la generación de la señal resultante en la versión definitiva del sintetizador FM.

4.3.2.1. Instrumento 1

Para el instrumento 1, hemos introducido dos novedades con respecto al sistema inicial: el LFO y los selectores de forma de onda. Dado que ya se estudió la implementación del prototipo, analizaremos únicamente las nuevas funcionalidades.

Con respecto al LFO, lo primero que debemos hacer es leer, mediante el *opcode* `invalue`, si el usuario ha activado el LFO a partir de un *checkbox* (`lfo_on`) y qué modo de modulación ha elegido (*trémolo* o *vibrato*), haciendo uso del *widget menu*, que ofrecerá una lista desplegable con ambas opciones (`menulfo`).

El estado de activación del LFO se almacenará en una variable de control que se denominará `klfo_on`. Por otro lado, cada opción del *widget menu* se hace corresponder en *QuteCsound* con un índice entero, que parte del cero y aumenta una unidad por cada nueva opción añadida al menú.

Así pues, al leer del *widget* `menulfo`, almacenaremos la opción escogida en la variable `kmenulfo`, de modo que si `kmenulfo==0`, habrá sido elegida la opción *Trémolo*, mientras que si `kmenulfo==1`, habremos optado por la opción *Vibrato*.

Proseguimos leyendo los valores de amplitud y frecuencia de dicho LFO a partir de los *knobs* `lfo_amp` y `lfo_frec`, respectivamente. Dichos valores los guardaremos en dos variables de tipo `k`: `klfoamp` y `klfofrec`.

Tras crear la señal moduladora según se vio en el prototipo, debemos generar la señal correspondiente al oscilador de baja frecuencia. En primer lugar, definimos una envolvente de tipo unidad ASR para el LFO, con tramos de ataque y relajación muy cortos, ya que nos interesa que la modulación en baja frecuencia sea "instantánea".

Para ello, empleamos el *opcode* `linenr`, especializado en la generación de envolventes en tiempo real, ya que la modulación en baja frecuencia se llevará a cabo durante el tiempo que tengamos pulsada la nota en el controlador MIDI.

El *opcode* `linenr` nos permite conseguir una envolvente de tipo ASR, introduciéndole como parámetros de entrada la amplitud de sostenimiento (`klfoamp`), el tiempo de ataque (en nuestro caso `0.01`, o 10 ms), el tiempo de relajación (también `0.01`) y un factor de atenuación que permite variar la curvatura del tramo de relajación entre lineal y exponencial negativa (por buenos resultados hemos elegido `0.001`).

Creada ya la envolvente del LFO, la almacenamos en la variable de tipo control `kenvlfo`.

A continuación, determinamos la señal de salida del LFO, haciendo uso de otro nuevo *opcode* que se denomina `lfo`. Funciona igual que `oscil` u `oscil3`, con la diferencia de que no es necesario crear una tabla de onda para generar la señal. En lugar de añadir el número de la tabla de onda, tendremos que introducir un número comprendido entre `0` y `5` (`itype`).

Cada número se corresponde con una forma de onda diferente (`0`: sinusoidal, `1`: triangular,...etc), de forma que nos ahorra la sentencia `f` en la partitura. En el caso que nos ocupa, necesitamos una señal sinusoidal, con lo que `itype = 0`.

Mediante la instrucción `klfo lfo kenvlfo, klfofrec, 0` implementaríamos la señal sinusoidal de tipo control indicada por el usuario (`klfo`), que posteriormente realizará una modulación en baja frecuencia sobre la señal portadora.

Ahora debemos discriminar mediante un condicional simple (`if`) si el usuario ha encendido el LFO. Si no lo ha encendido, es decir, `klfo_on==0`, crearíamos la señal portadora modulada en FM (`aport`) como si se tratara del prototipo, pero teniendo en cuenta el selector de forma onda (más adelante profundizaremos en ésto).

Por otro lado, si se encuentra encendido el LFO (`klfo_on==1`), tenemos dos posibilidades: la función *Trémolo* está seleccionada (`kmenulfo==0`) o bien ha sido escogida la función *Vibrato* (`kmenulfo==1`).

Si nos encontramos en el primer caso, la amplitud de la portadora tendría que ser modulada según `klfo`, por lo que valdría: `iamp*(apenv+a(klfo))`. En la anterior expresión, podemos fijarnos en dos aspectos:

- La variable `iamp` se ha puesto fuera multiplicando al término (`apenv+a(klfo)`) para que la modulación en baja frecuencia también sea proporcional a la amplitud de la nota MIDI pulsada.
- Se ha realizado *casting* en la variable `klfo`, convirtiéndola a tipo audio, ya que el resultado final de amplitud será posteriormente escuchado. Además no mezclamos distintos tipos de variables.

Cuando estemos en el segundo caso, la señal `klfo` actuará como modulador en frecuencia de la señal portadora `aport`. En éste sentido, la expresión que deberemos introducirle como parámetro de frecuencia será: `(ifrec*knport+amod)*(1+a(klfo))`. En dicha expresión destacamos lo si-

guiente:

- El término `ifrec*knport+amod` hace referencia a la modulación FM simple.
- Puesto que `klfo` variará entre -1 y 1, la frecuencia de la portadora variará entre `0` y `2*(ifrec*knport+amod)`.
- Se podría realizar la modulación del LFO con la expresión `(ifrec*knport+amod)*a(klfo)`, pero el resultado será menos perceptible que con la expresión utilizada.

Por último, al haber generado la señal portadora `aport`, ya podemos transferirla a la variable global de audio (`gabus`), esto es, `gabus = gabus + aport`.

La segunda y última novedad que le hemos introducido al instrumento `instr 1`, es el selector de forma de onda para cada oscilador.

Empezamos creando dos *widgets* de tipo *menu* que ofrecerán un lista desplegable con cinco opciones diferentes, una para cada forma de onda: `Sinusoidal`, `Cuadrada`, `Triangular`, `Diente de sierra` y `Pulso`.

Los *widgets* recibirán como nombre de canal `menufport` y `menufmod`. Además, crearemos dos *widgets* adicionales de tipo *graph*, que se llamarán igual que los menús desplegables y su función será representar la forma de onda seleccionada para cada señal.

Seguidamente, en el código del cuerpo del instrumento 1, procedemos a la lectura de las opciones escogidas para cada menú, mediante `invalue`. Ambas opciones se almacenan en las variables `kptabla` (índice de forma de onda de portadora) y `kmtabla` (índice de forma de onda de moduladora).

La idea de dichos índices reside en que sean capaces de controlar qué tabla (creada previamente en la partitura) se enlazará como tabla de onda del *opcode* `oscil3` correspondiente.

Para ello, lo que hacemos es introducir como argumento de tabla de onda del `oscil3` referente al modulador, el valor `i(kmtabla+1)`, y para el portador, `i(kptabla+1)`.

Recordemos que el *opcode* `oscil3` admite como número de tabla de onda, una variable de tipo nota (`i`), con lo que debemos convertir la operación resultante de tipo `k`. Por otro lado, sumamos una unidad a `kmtabla` y `kptabla` puesto que en la partitura no podemos poner como

identificadores de tablas de onda los mismos índices de los menús, ya que no está permitido el uso del cero como identificador de tabla.

Con lo cual, nosotros hemos creado la siguientes tablas de onda, empezando por el identificador `1` y acabando por el `5`:

```
;Ciclo de onda sinusoidal.
f 1 0 4096 10 1
;Ciclo de onda cuadrada con 8 armónicos.
f 2 0 4096 10 1 0 [1/3] 0 [1/5] 0 [1/7] 0 [1/9] 0 [1/11] 0 [1/13] 0 [1/15]
;Ciclo de onda triangular con 8 armónicos.
f 3 0 4096 10 1 0 -[1/9] 0 [1/25] 0 -[1/49] 0 [1/81] 0 -[1/121] 0 [1/169] 0 -[1/225]
;Ciclo de onda diente de sierra con 8 armónicos.
f 4 0 4096 10 1 [1/2] [1/3] [1/4] [1/5] [1/6] [1/7] [1/8]
;Ciclo de onda pulso con 8 armónicos.
f 5 0 4096 10 1 1 1 1 1 1 1 1
```

De éste modo, cuando seleccionemos desde el menú de portadora la opción `Sinusoidal` (por ejemplo), la variable `kptabla` tomará el valor `0`, pero al sumarle una unidad, valdrá `1`, con lo que invocaremos desde el `oscil3` a la tabla de onda cuyo identificador es `1`, es decir, la sinusoidal.

En la Fig. 4.45 se encuentra el diagrama de bloques detallado que muestra la implementación comentada del instrumento `instr 1`, referente a la versión final del sistema.

4.3.2.2. Instrumento 100

La misión principal del instrumento `instr 100` será permitir (a decisión del usuario) la introducción de *delay* en la señal de salida (`gabus`), una vez procesada por el instrumento `instr 1`.

Para ello, comenzamos obteniendo el valor de la casilla de activación del módulo de *delay* (`delay_on`), utilizando el *opcode* `invalue`, y almacenamos el estado de dicha casilla en una variable de tipo control denominada `kdelayon`.

Mediante un condicional simple, comprobamos si se ha encendido el dispositivo de *delay*. Si estuviera apagado (`kdelayon==0`), el instrumento finalizaría su ejecución.

En caso de estar encendido (`kdelayon==1`), guardamos los valores que ha introducido el usuario de tiempo de retardo, realimentación y *dry/wet*, en las variables `ktiempo`, `krealim` y `kdrywet`, respectivamente. Los *knobs* dedicados a dichos parámetros serían `delay_t`, `delay_r` y `delay_dw`.

Cuando ya disponemos de los parámetros necesarios para construir el

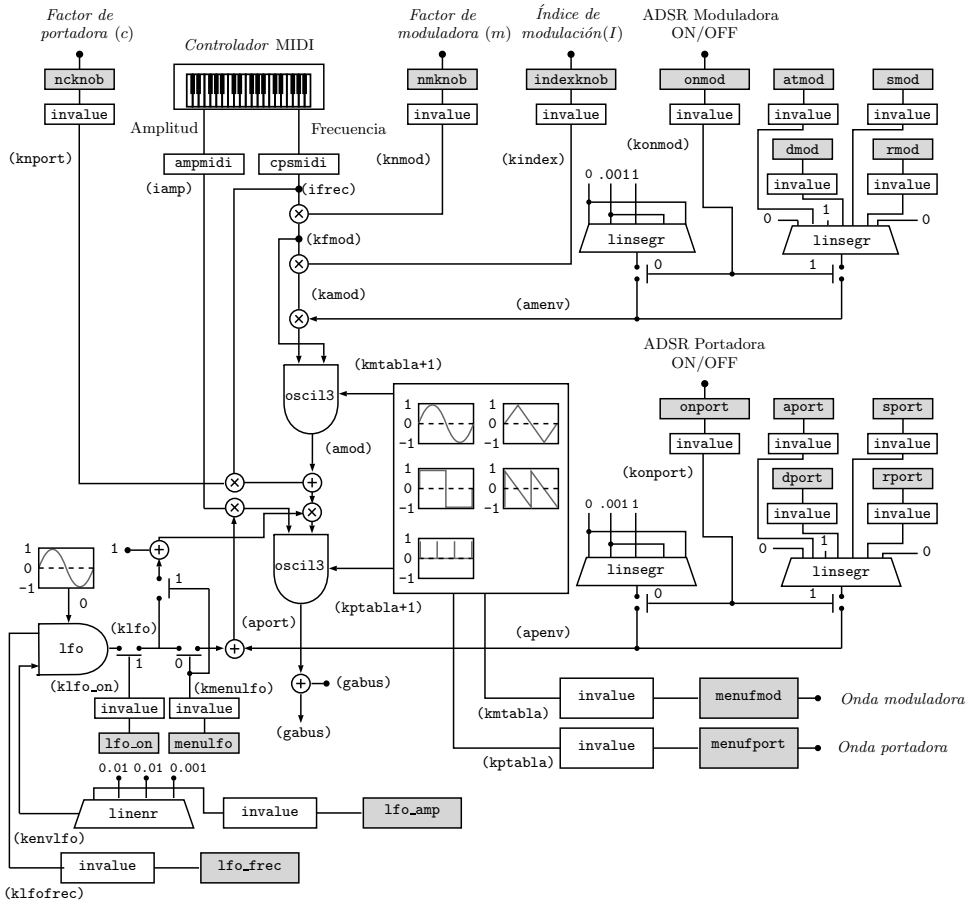


Figura 4.45: Diagrama de bloques detallado del instrumento `instr 1`.

delay, creamos una línea de retardo de 1 segundo mediante el *opcode* `delayr`, que se corresponde con el máximo tiempo de retardo que se podrá configurar para éste efecto, y la almacenamos en una variable de tipo audio que nombraremos `adelay`.

El siguiente paso, consiste en definir la variable que se encargará de almacenar la señal retardada `ktiempo` segundos (`awet`), utilizando el *opcode* `deltapi`.

El *opcode* `delayw` hará posible la escritura del sonido en la línea de retardo. No devuelve ninguna variable de salida y como entrada tomará la expresión: `gabus + (awet*krealim)`. Este *opcode* se utiliza siempre en conjunto con `delayr`.

El último paso que se realiza es añadir a la variable global de audio

una mezcla de la señal *seca* y la señal retardada, dependiendo del valor que se haya escogido para el *dry/wet*: `gabus = (1-kdrywet)*gabus + kdrywet*awet`.

Para ver el proceso de una forma gráfica, observe el diagrama detallado de éste instrumento (Fig. 4.46).

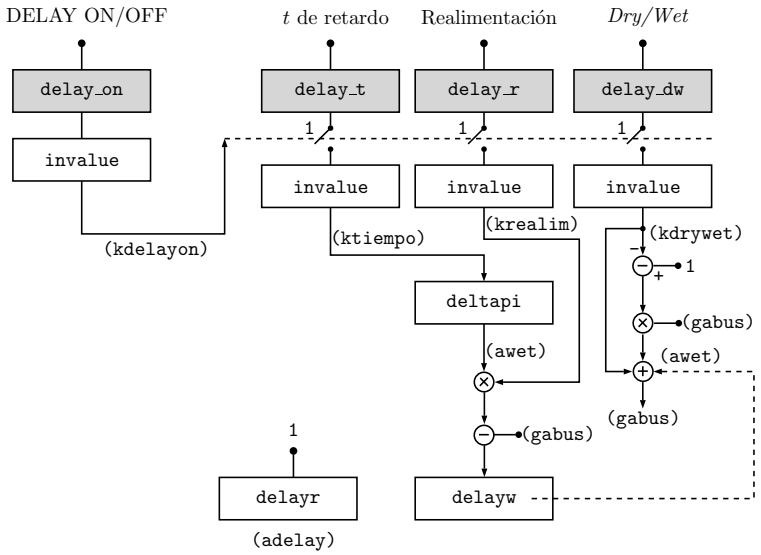


Figura 4.46: Diagrama de bloques detallado del instrumento `instr 100`.

4.3.2.3. Instrumento 101

El instrumento `instr 101` comienza su definición leyendo (como siempre, con `invalue`) el valor del *checkbox* de activación del *reverb* (`rev_on`). Dicho valor se almacena en una variable de tipo `k`: `krev_on`.

Procedemos, a continuación, de la misma forma que en el instrumento anterior, es decir, a través de un condicional simple verificamos si el usuario ha activado o desactivado la casilla de encendido del *reverb*.

Si el usuario no ha activado la casilla, el *reverb* estará apagado, por lo que la variable `rev_on` valdrá `0` y la ejecución del instrumento finalizaría.

Sin embargo, cuando el usuario active ésta casilla, la variable `rev_on` tomará el valor `1`, accediendo al bloque del condicional y permitiendo así realizar el procesado de la reverberación a la señal de salida.

En la interfaz gráfica tendremos tres *widgets* de tipo *knob*, dedicados

a recoger los valores del factor de tamaño de sala, la atenuación de altas frecuencias y la proporción *dry/wet*.

Cada parámetro registrado a partir de éstos *knobs* mediante `invalue`, se almacenará en las variables de tipo control `ktam`, `kataf` y `kdrywet`.

El *opcode* que nos permitirá implementar la reverberación será `freeverb`. Se trata de un *opcode* que aplica reverberación estéreo a una señal de entrada que presente dos canales: izquierdo (L) y derecho (R).

En nuestro caso, como la variable global de audio `gabus` es monofónica, introduciremos como parámetros de entrada de `freeverb` la misma señal repetida, para el canal L y R. Después, se introducen los parámetros característicos de la reverberación: `ktam` y `kataf`.

Por último, obtenemos a la salida del *opcode* la señal de audio con reverberación aplicada: `awet`. Nótese que se devuelve dos veces, ya que la salida del *opcode* es estéreo, aunque se trata de la misma señal duplicada.

Solo queda añadir el control de *dry/wet*. Para ello, actualizamos el valor de `gabus`, sustituyéndolo por la proporción de señal con y sin reverberación: `gabus = (1-kdrywet)*gabus + kdrywet*awet`. Recordemos que como `kdrywet` varía entre `0` y `1`, cuando sea nula (`kdrywet==0`) solo tendremos señal seca; mientras que cuando valga la unidad (`kdrywet==1`) únicamente dispondremos de señal reverberada.

La implementación detallada de éste instrumento se puede estudiar en el diagrama adjunto:

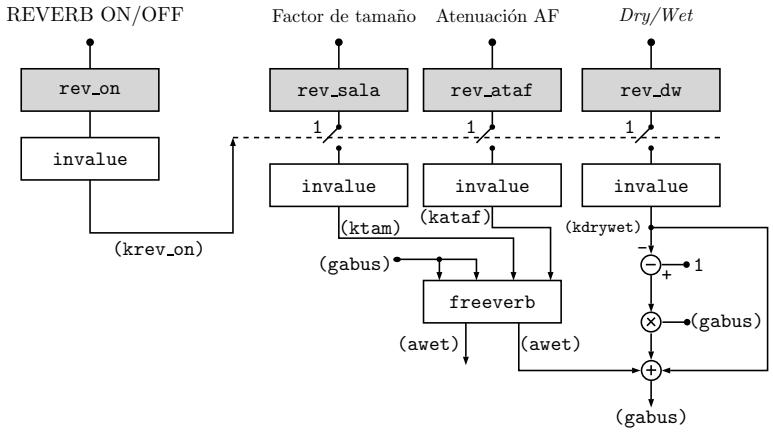


Figura 4.47: Diagrama de bloques detallado del instrumento `instr 101`.

4.3.2.4. Instrumento 102

El instrumento `instr 102` se encarga de gestionar los detectores (ON, nota-ON MIDI y CLIP), así como amplificar la señal de salida y representar su espectro en tiempo real. Además, también se encarga de llevar el bus de audio a la tarjeta de sonido, e inicializarlo a cero para que no se produzca realimentación positiva.

La definición de éste instrumento comienza mediante la lectura del estado del botón de apagado del sintetizador (`_Stop`), almacenándolo en la variable `koff`. Como vemos, éste nombre responde a un canal reservado de *QuteCsound*, y sirve para detener la ejecución del programa.

Mediante un condicional, discriminamos dos situaciones:

- No se ha pulsado el botón OFF del sintetizador y nos encontramos en la primera iteración del bucle correspondiente al bus de sonido.
- Sí se ha presionado el botón OFF.

Si nos encontramos en el primer caso querrá decir que `koff==0&&giaux==0`. Cabe destacar, que `giaux` es una variable global auxiliar, definida en la cabecera de la orquesta e inicializada a `0` con el *opcode* `init` (`giaux init 0`), de igual forma que `gabus`.

La variable `giaux` servirá como centinela para saber si nos encontramos en la primera iteración o no. De modo que antes de entrar en el condicional simple, su valor será `0`, pero cuando vaya a salir de él, tomará el valor `1`. Con ésto ya sabremos que la primera iteración ha sucedido.

Retomando el primer caso del condicional (botón OFF sin pulsar y primera iteración), procedemos a encender el *display* ON del sintetizador, que indicará que se encuentra encendido.

La manera de hacer ésto es enviar a un *widget* de tipo *display* (denominado `ondisp`) una cadena de texto que únicamente contenga el carácter punto (`.`). Así, desde el panel de *widgets* podemos modificar el formato del *display* `ondisp`, añadiéndole color verde intenso para simular un LED.

Para mandar el punto de texto al *display*, lo haríamos a través del *opcode* `outvalue`, mediante la siguiente sentencia: `outvalue "ondisp","."`.

A continuación, asignamos a la variable centinela `giaux` el valor `1`. El hecho de que nos interese saber si nos encontramos en la primera iteración o no, es para no encender el LED constantemente ya que ésto

significaría un procesado innecesario. Por consiguiente, solo necesitamos encender el LED en la primera iteración, apagándose únicamente cuando finalice la ejecución del sintetizador.

En el segundo caso del condicional, si se ha pulsado el botón OFF (`koff==1`), apagaríamos todos los *displays* que hay presentes en el dispositivo (`ondisp`, `clip` y `notaon`), transmitiéndoles con `outvalue` una cadena de texto vacía. Finalmente, pondríamos a cero el bus de audio principal (`gabus=0`), con lo que terminaría el condicional del detector ON.

En éste momento, obtenemos información de la nota MIDI que estamos pulsando desde el controlador, empleando el *opcode* `midiin`. Se encarga de mandarnos información sobre el estado del mensaje MIDI recibido (`kstatus`), el canal MIDI (`kchan`) y dos datos (`kdata1` y `kdata2`), que dependerán del tipo de mensaje MIDI.

En el caso de un mensaje MIDI de tipo nota-ON, `kdata1` haría referencia a la nota y `kdata2` se correspondería con la velocidad.

No obstante, `midiin` no recibe ningún parámetro, ya que la lectura del puerto MIDI la realiza de manera implícita.

Seguidamente, implementamos otro condicional `if`, en el que si se ha recibido un mensaje de nota-ON (`kstatus==144`) encendemos el *display* `notaon` con `outvalue` (color naranja). En el caso en que recibamos un mensaje de nota-OFF (`kstatus==128`), apagaríamos dicho *display*.

Es frecuente que algunos controladores MIDI no permitan enviar mensajes de nota-OFF[8]. Si ésto sucediera, siempre podemos considerar que un mensaje nota-OFF es como un mensaje de nota-ON con velocidad nula, por lo que sustituiríamos la condición `kstatus==128` por `kdata2==0`.

Ahora, pasaríamos a la lectura del potenciómetro de ganancia (`nivel`) para ajustar el MASTER. El valor registrado por `invalue` lo guardaríamos en la variable `knivel`.

Proseguimos con la definición de una nueva variable de tipo audio (`asalida`), que será el resultado de aplicar el factor de ganancia `knivel` a la variable global de audio: `asalida = knivel*gabus`.

Para construir el detector de CLIP o distorsión, determinamos el máximo

[8]Para conocer información sobre el tipo de mensajes que puede enviar el controlador utilizado, véase la tarjeta de implementación MIDI en el manual del usuario del dispositivo.

o mínimo local de la señal sintetizada cada 0,1 segundos y si dicho valor es mayor o igual a la unidad, encendemos el *display*. Ésto lo llevaremos a cabo mediante el *opcode* `max_k`.

`max_k` determina el máximo o mínimo local de la señal de entrada `asig`, en el intervalo de tiempo en que la señal de reloj `ktrig` toma el valor `1` dos veces consecutivas. Además, es necesario especificar un tercer parámetro de entrada (`itype`), que indica si queremos obtener el máximo o mínimo en valor absoluto (`itype=1`), solo el máximo (`itype=2`), solo el mínimo (`itype=3`) o un promedio de la señal (`itype=4`).

Antes de utilizar `max_k`, tenemos que definir la señal de reloj, empleando el *opcode* `metro`. Éste genera una señal de reloj asíncrona `ktrig` de frecuencia `kfreq`.

Por ello, generamos nuestra señal de reloj `ktrigclip` de frecuencia `10` Hz (periodo `0,1` segundos): `ktrigclip metro 10`.

A continuación, obtenemos el máximo o mínimo en valor absoluto de la señal (`kclip`) según el periodo de `ktrigclip`, con lo que: `kclip max_k asalida, ktrigclip, 1`.

Discriminando con un condicional simple, encendemos el *display* `clip` (mandándole un carácter punto) en caso de que `kclip` sea igual o superior a la unidad; y lo apagamos en caso contrario (`kclip<1`).

Transmitimos la señal `asalida` a la salida de la tarjeta de audio mediante el *opcode* `outs`, equivalente al `out`, pero en estereofónico. En éste momento, ya estaríamos escuchando la señal final sintetizada que ha pasado por los sucesivos instrumentos.

El siguiente paso consiste en averiguar si el usuario ha activado la casilla de representación del espectro. Por tanto, leemos el estado del *widget* `espectro_on` y guardamos su valor en `kespectro`.

Si el usuario la ha activado (`kespectro==1`), procedemos a la representación del espectro en tiempo real con el *opcode* `dispfft`.

Su función es la de representar el espectro de una señal de tipo `a` o `k` (`xsig`), con un periodo de representación (en segundos) `iprd` y un tamaño de ventana de procesado (en muestras) `iwsiz`.

Los parámetros de `dispfft` dependen en gran medida de la capacidad del ordenador que en el que se esté ejecutando el programa, por lo que

puede ser necesaria una reconfiguración. Hemos obtenido una representación fluida para `iprd = 0.01` y `iwsiz = 256`. El comando que realizaría la representación del espectro de `asalida` sería: `dispfft asalida, 0.01, 256`.

Para representar el espectro que acabamos de calcular, enviamos al *widget* `espectro` el índice `5`, que se corresponde con el número total de tablas que tenemos creadas. Ésto se debe a que *QuteCsound* también detecta las tablas de onda por sus índices.

En éste sentido, cuando creamos la primera tabla de la partitura, *QuteCsound* le asigna el índice `0`, para la segunda el índice `1`,...etc. Como nosotros hemos creado 5 tablas, la última tendrá el índice `4` con lo que el siguiente índice libre al último creado sería el `5`. Por eso añadimos un `5` al `outvalue` de `espectro`.

Finalizamos el instrumento estableciendo la variable global del bus de audio a cero (`gabus=0`), con lo que evitamos que se acumulen los valores de todas las iteraciones anteriores y por consiguiente, se produzca realimentación positiva en el sonido.

El flujo de información de éste último instrumento se refleja en la Fig. 4.48. Si desea más información sobre la implementación, revise el Anexo A de la presente memoria, donde figura el código fuente del proyecto.

4.3.3. Interfaz gráfica definitiva

La interfaz gráfica propuesta para la versión final del sistema sería la que aparece en la Fig. 4.49. Podemos destacar que se han añadido 4 nuevos módulos, siguiendo el formato *rack* de la versión inicial.

El primer módulo engloba todos los *displays* de los detectores estudiados, el potenciómetro de MASTER, los botones de encendido y apagado del sistema y el selector de *presets*.

En lo que respecta al efecto gráfico de los *displays* para emular los LED, podemos destacar que ha sido necesario, para cada detector, un *label* de fondo y el *display* propiamente dicho.

El *display* ya sabemos que recibirá un carácter punto, y desde el modo edición del panel de *widgets*, le daremos un color muy vivo para simular que está encendido si se activa el detector.

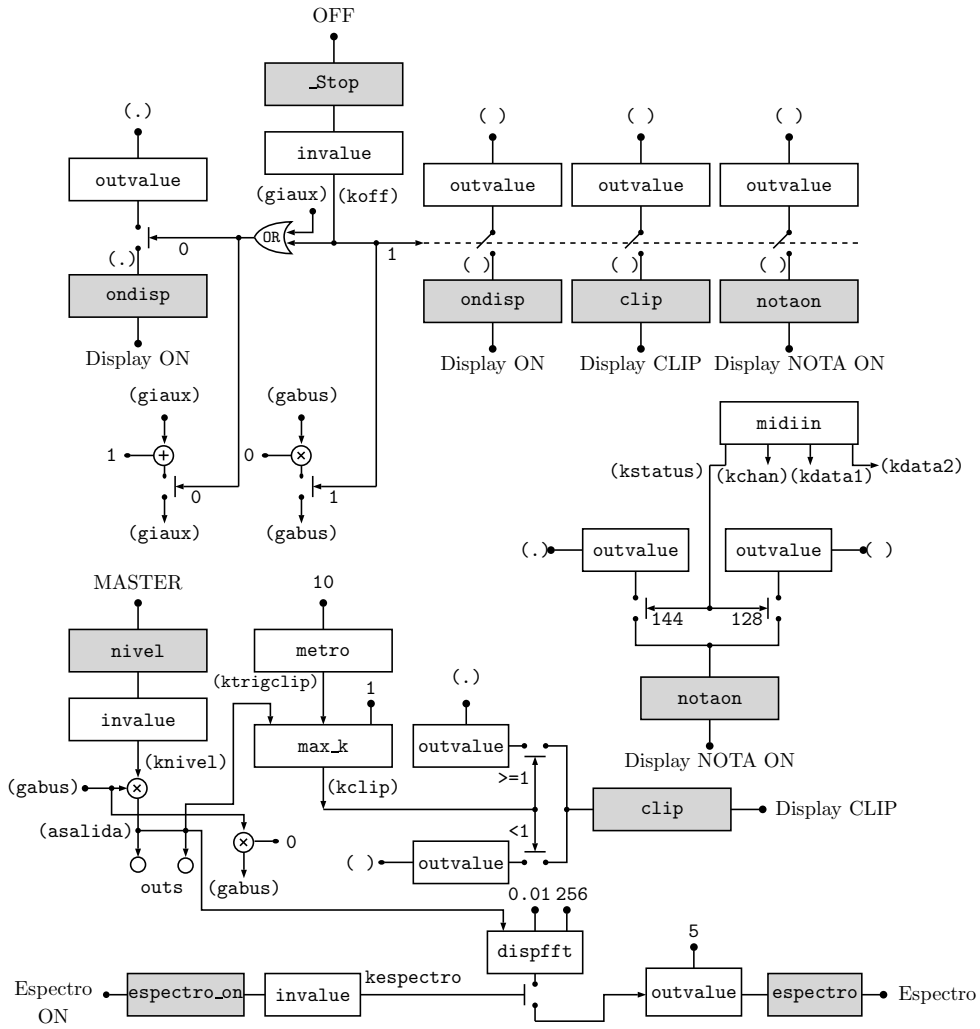


Figura 4.48: Diagrama de bloques detallado del instrumento `instr 102`.

En cambio, el *label* deberá estar situado en la misma posición del *display* pero un plano inferior, y le añadiremos el mismo color que al *display* correspondiente, aunque con una mayor proporción de negro. Así parecerá que el detector está apagado cuando no se envíe ningún carácter al *display*.

El *knob* correspondiente al potenciómetro de MASTER (`knivel`) se ha configurado para variar en un rango entre `0` y `1`.

Una de las cosas más interesantes de éste módulo de la interfaz es el selector de *presets*. En primer lugar, hemos creado dos *widgets*: un *spinbox* y un *display*.

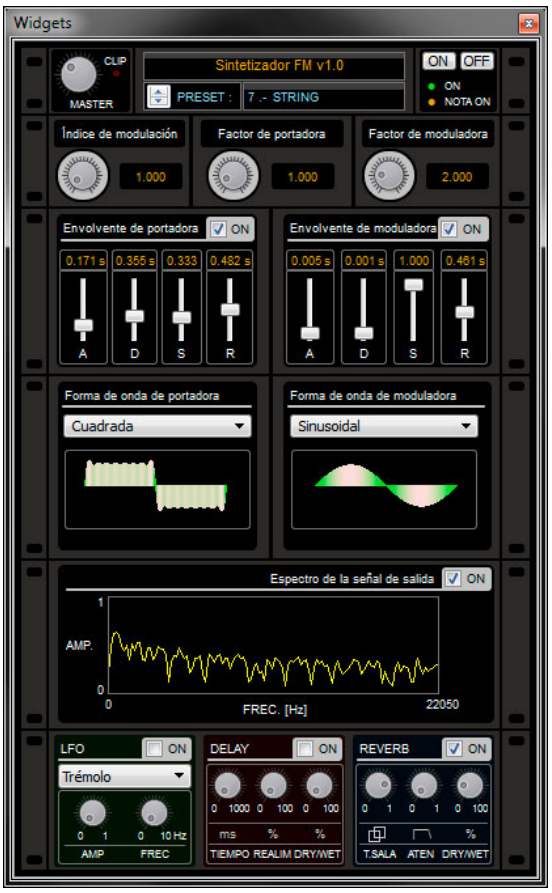


Figura 4.49: Interfaz gráfica de la versión definitiva del sintetizador FM.

El *spinbox* tomará como nombre de canal `_SetPreset`, que se trata de un canal reservado para variar el número del *preset*, y por tanto, cargar el *preset* correspondiente en el sintetizador. Debe tomar como valor mínimo `1`, y como máximo, el número máximo de *presets* que deseemos añadir, inicialmente `9`.

El *display* tomará el mismo nombre que el *spinbox*, es decir, `_SetPreset`. Puede parecer extraño que a un *widget* de salida le estemos asignando un canal reservado de entrada. Ésto se debe a que el papel del *display* es mostrar únicamente el número correspondiente al *preset* cargado, y no modificarlo.

Lo último que queda es añadir otro *display* más, pero ésta vez con el nombre del canal reservado `_GetPresetName`. La función de éste *widget* será obtener el nombre del *preset* cargado.

Los botones ON y OFF ya sabemos que activan y desactivan el sintetizador. Para ello, el botón ON recibe como nombre de canal `_Play` (inicia la ejecución), mientras que el botón OFF toma como nombre el canal reservado `_Stop` (detiene la ejecución). Ambos botones se han configurado desde el panel de propiedades con el parámetro `Type` en modo `value`.

Los selectores de formas de onda para las señales portadora y moduladora se corresponden con el segundo nuevo módulo. Para cada selector, hemos creado un *widget menu* y otro *graph*. Los menús (`menufport` y `menufmod`) tienen como opciones seleccionables los nombres de las 5 formas de onda.

Por otro lado, los *graph* nos permitirán representar las tablas referentes a las formas de onda seleccionadas por ambos menús. Para ello, tenemos que darles como nombre de canal el mismo que el de sus respectivos menús. Para reducir las formas de onda de los *graph*, hemos establecido la escala del eje X a `0.8` y la del eje Y a `0.5`.

El tercer nuevo módulo es el de representación espectral. Su implementación se ha llevado a cabo mediante dos *widgets*: un *checkbox* y un *graph*.

El *checkbox* (`espectro_on`) tomará el valor `1` cuando se marque, iniciando la representación del espectro en el *graph* (`espectro`). La escala utilizada en el *graph* ha sido `0.8` para ambos ejes de coordenadas.

El cuarto y último módulo es el de LFO, *delay* y *reverb*.

El LFO emplea un *checkbox* (`lfo_on`), que activa el LFO cuando se marca (`1`); un *menu* (`menulfo`) con las opciones *Trémolo* y *Vibrato*; y dos *knobs* para controlar la amplitud y frecuencia del LFO.

El primero de los *knobs* tiene como nombre de canal `lfo_amp` y varía entre `0` y `1` (amplitud máxima y mínima de la modulación en baja frecuencia). El segundo se llama `lfo_frec` y se ha configurado para que la frecuencia de la modulación se pueda modificar entre `0` y `10` Hz.

El dispositivo correspondiente al *delay* contiene un *widget checkbox* que se denomina `delay_on` para activar o desactivar el efecto, según tome el valor `1` o `0`, respectivamente. Finalmente, los tres *knobs* referentes al tiempo de retardo, realimentación y *dry/wet* (`delay_t`, `delay_r` y `delay_dw`), se han limitado a los valores `0` y `1`.

El último bloque sería el de reverberación y se compone por el mismo

número y tipos de *widget* que el módulo de *delay*. Los *knobs* correspondientes a la atenuación de alta frecuencia y el control de *dry/wet* (`rev_ataf` y `rev_dw`) variarán entre `0` y `1`, mientras que el *knob* del factor de tamaño de sala (`rev_sala`) se podrá utilizar entre `0` y `0.95`, puesto que valores cercanos a `1` pueden hacer inestable al *opcode* `freeverb`.

4.3.4. Experimentación y validación

En éste apartado, experimentaremos con los nuevos módulos añadidos, realizando diversas pruebas para comprobar que el sistema responde según las exigencias planteadas.

4.3.4.1. Selectores de forma de onda

Para saber si los selectores de forma de onda funcionan de modo apropiado, comenzaremos verificando el selector de la portadora, configurando primero el factor de portadora a 1. Tanto el índice de modulación como el factor de moduladora los fijaremos en 0, para que no se produzca modulación FM.

Ahora, ejecutamos una nota en el controlador MIDI, y vamos seleccionando cada una de las cinco formas de onda que ofrece el menú:

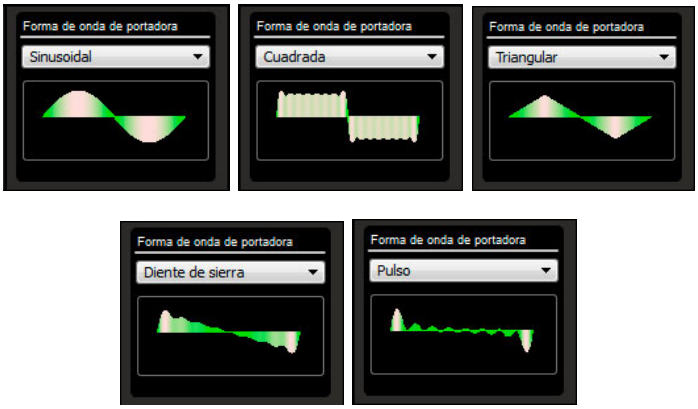


Figura 4.50: Tipos de forma de onda que ofrecen los selectores.

Escuchemos en los siguientes archivos los resultados obtenidos:

22.-Sinusoidal.mp3 23.-Cuadrada.mp3 24.-Triangular.mp3

25.-Diente de sierra.mp3 26.-Pulso.mp3

Podemos observar que hemos conseguido los resultados esperados. No obstante, como únicamente hemos utilizado 8 armónicos para crear las distintas ondas (salvo la sinusoidal que solo contiene uno), hemos obtenido aproximaciones de las ondas teóricas que presentan infinitos armónicos.

Las siguientes figuras muestran las formas de onda cuadrada y pulso, representadas en el dominio del tiempo:

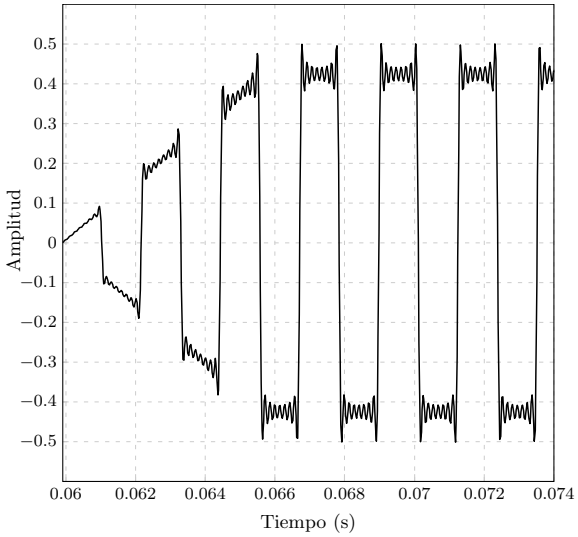


Figura 4.51: Representación del resultado obtenido para la forma de onda cuadrada.

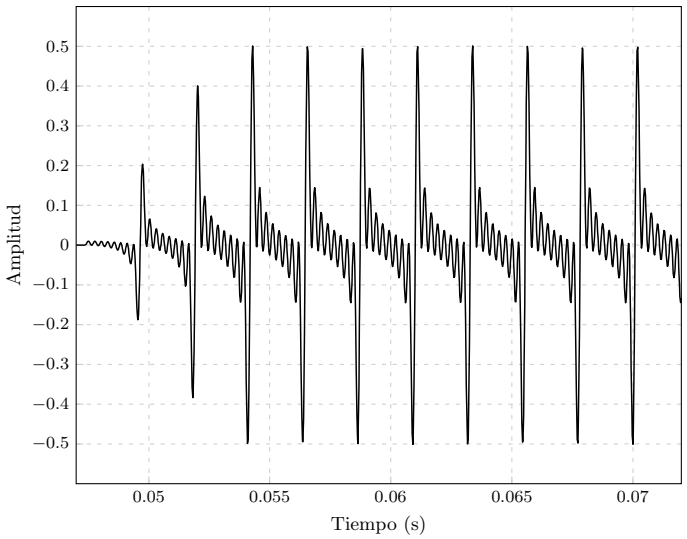


Figura 4.52: Representación del resultado obtenido para la forma de onda pulso.

Para verificar el selector de forma de onda de moduladora, hemos realizado previamente una modulación, seleccionando como forma de onda de portadora una señal cuadrada, y para la moduladora una triangular:

Figura 4.53: Parámetros de ajuste escogidos para comprobar el selector de onda de la moduladora.

Escuchemos seguidamente el resultado:

27.-Modulación de cuadrada con triangular.mp3

Podemos apreciar que la modulación sobre la señal cuadrada se produce de un modo progresivo debido al valor de ataque de la envolvente de la señal moduladora (1 s).

Acto seguido, se distingue una ligera pérdida de parciales debido al tramo de decaimiento, durante 0,834 s.

Finalmente, al estabilizarse en la fase de sostenimiento, podemos oír la modulación que provoca la señal triangular sobre la cuadrada. Nótese que existe un mayor número de armónicos en ésta modulación que si moduláramos con una sinusoidal, por lo que el sonido será más brillante.

Por consiguiente, podemos concluir que los selectores de forma de onda funcionan según lo previsto.

4.3.4.2. Representación del espectro

En éste apartado, propondremos tres experiencias para asegurar que el módulo referente a la representación del espectro en tiempo real responde de forma adecuada.

Vamos a ejecutar una nota sin aplicar modulación ($I = 0$ y $m = 0$), escogiendo para c un valor de 1, con lo que la frecuencia de la nota pulsada se corresponderá con la emitida.

Activamos una envolvente ADSR para la portadora y emulamos una envolvente tipo unidad, para tener tiempos muy cortos de ataque, decaimiento y relajación, así como para evitar *clicks* en el inicio y final del sonido.

El siguiente paso consiste en seleccionar una forma de onda de portadora de tipo sinusoidal, activar el módulo de espectro y pulsar la nota. El resultado sería el que se muestra en la siguiente figura:

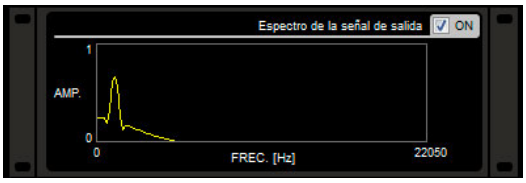


Figura 4.54: Representación del espectro en tiempo real de una señal sinusoidal.

Como vemos, el espectro representa la amplitud de la señal normalizada frente a la frecuencia en escala lineal. La máxima frecuencia es de 22050 Hz (`sr/2`), de acuerdo con el *Teorema de Nyquist*.

Es fácil distinguir que los resultados son los esperados, puesto que al tratarse de una función sinusoidal, solo existirá un único parcial en la señal de salida, lo cual se refleja en la figura anterior.

Si realizamos la misma experiencia pero cambiando ésta vez la señal sinusoidal por una cuadrada, tendríamos la siguiente representación:

Figura 4.55: Representación del espectro en tiempo real de una señal cuadrada con 8 armónicos.

La señal se estaría representando correctamente, ya que contiene 8 armónicos impares cuyas amplitudes decaen de forma proporcional a $1/n$, donde n es el índice del parcial.

Finalmente, podemos ver el espectro que resulta de una modulación FM, partiendo de una portadora triangular y una moduladora diente de sierra.

Figura 4.56: Representación del espectro en tiempo real para un ejemplo de señal con modulación FM.

El espectro gana complejidad armónica puesto que al modular una señal triangular (que contiene 8 parciales) con una triangular (también de 8 parciales) nos encontraríamos en una situación de FM compuesta: 8 moduladoras que modulan a 8 portadoras.

4.3.4.3. LFO, Delay y Reverb

Para verificar el LFO, vamos a probar cuatro combinaciones posibles de amplitud y frecuencia, tanto para modo *Trémolo* como *Vibrato*.

Comenzaremos con amplitud y frecuencia bajas, proseguiremos con amplitud baja y frecuencia alta; después con amplitud alta y frecuencia baja y finalmente, con amplitud y frecuencia altas.

Empecemos con el modo *Trémolo*, para una señal portadora de tipo cuadrada (Fig. 4.57).

Si accedemos al siguiente fichero de audio, podremos apreciar que la modulación en baja frecuencia de la amplitud de la portadora se realiza correctamente para los 4 casos. Como vemos, la amplitud del LFO en modo

Figura 4.57: Diversas configuraciones para verificar el LFO en modo *Trémolo*.

28.-LFO-Trémolo.mp3

Trémolo determina cuál será la desviación máxima y mínima de la amplitud de la portadora. En cambio, la frecuencia del LFO nos indicará cómo de rápido se desviará dicha amplitud.

Para el caso del *Vibrato* procederíamos de la misma forma, realizando en cada ocasión los ajustes que se muestran en la figura 4.58, también para una señal portadora de tipo cuadrada.

Figura 4.58: Diversas configuraciones para verificar el LFO en modo *Vibrato*.

La amplitud y frecuencia del LFO nos indicaría (en el *Vibrato*) en qué rango y a qué velocidad variará la frecuencia de la señal portadora.

El resultado puede oírse en el siguiente fichero, alojado en el directorio `Audio`:

29.-LFO-Vibrato.mp3

Puesto que los resultados son satisfactorios, podemos pasar al análisis de los efectos digitales.

Para comprobar que el *delay* se aplica correctamente a la señal de salida tras activarlo, hemos realizado cuatro experiencias.

En la primera experiencia, hemos asignado un tiempo de retardo menor a 300 ms y un factor de realimentación inferior al 30 % (manteniendo el *dry/wet* fijado al 30 %), por lo que el intervalo de tiempo entre la señal

directa y la retardada no solo será muy corto, sino que la señal retardada se disipará muy pronto.

La siguiente prueba, es contraria a la primera, en el sentido en que aunque mantenemos la cantidad de efecto constante (*dry/wet* al 30 %), aumentamos el tiempo de retardo a 1 segundo y la realimentación al 80 % aproximadamente. En éste caso, apreciamos que el tiempo entre la señal directa y la retardada se triplica con respecto al anterior caso, y la señal retardada tarda mucho más en disiparse debido a la elevado *feedback*.

En la tercera y cuarta prueba comprobaremos el control del *dry/wet*. Para ello, ajustamos el tiempo de retardo a 200 ms, la realimentación al 20 % y el *dry/wet* a un valor bajo, entorno al 20 %. Con éstos parámetros, la señal se repetirá de modo muy frecuente y tardará muy poco en disiparse, aunque se percibirá la señal directa a mayor amplitud que la señal retardada (la cantidad de efecto es baja).

Por otro lado, si mantenemos los mismos ajustes, pero variamos el *dry/wet* ajustándolo en el 80 %, las características del retardo serán las mismas, pero al existir mayor proporción de señal retardada que de señal directa, notaremos que la señal retardada presenta mayor amplitud que la directa.

Figura 4.59: Diversas configuraciones para comprobar el módulo de DELAY.

Las cuatro pruebas se han agrupado en el siguiente fichero de audio:

30.-Delay.mp3

Para el módulo de reverberación, realizaremos 3 pruebas. Primeramente, ajustamos el factor del tamaño de sala a 0,2 y el factor de atenuación de altas frecuencias a cero, de modo que simularemos estar en una sala de tamaño reducido (habitación común) en la que las altas frecuencias se atenuarán al mismo ritmo que las medias y bajas. El *dry/wet* se ajustará alrededor del 20 %.

El segundo tipo de *reverb* que crearemos será con un factor de tamaño de sala de 0,8, factor de atenuación de altas frecuencias a 0,2 y control de *dry/wet* al 20 %. En éste caso, parecerá que el sonido directo se ha

emitido desde una sala de grandes dimensiones, atenuando las frecuencias elevadas de forma ligera.

La última prueba que realizaremos se basará en mantener los parámetros anteriores, modificando exclusivamente el *dry/wet* a un valor de 80 %, con tal de apreciar qué sucede en la señal de salida.

Analizando el sonido producido, podemos distinguir que existe una reverberación muy exagerada, que cambia por completo la señal original, añadiendo además un carácter metálico. Ésto se debe a que, con éste ajuste, apenas queda señal seca, con lo que únicamente escuchamos una amplificación de las reflexiones producidas en la sala emulada.

Figura 4.60: Diversas configuraciones para comprobar el módulo de REVERB.

Abra el fichero de audio en un reproductor para distinguir los tres tipos de reverberación producidos:

`31.-Reverb.mp3`

4.3.4.4. Detectores

El comportamiento del detector ON es correcto, puesto que al pulsar tanto el botón ON del sintetizador como el botón *Correr* de *QuteCsound*, ambos ejecutan el programa y encienden el *display* verde.

En cambio, al pulsar sobre el botón OFF del sintetizador, el *display* pasa de estado encendido a apagado (no se envía el carácter punto). Cabe destacar, además, que el botón OFF no apagaría únicamente el detector ON, sino también el de nota-ON MIDI y el de CLIP.

Un factor sorpresa de *QuteCsound* reside en que si pulsamos el botón *Parar*, el *display* no se apaga.

Ésto es debido a que el proceso mediante el que se detiene la ejecución de *QuteCsound* con el botón *Parar* no es el mismo que con el que se detiene mediante el botón OFF, que invoca al canal reservado `_Stop`.

Desgraciadamente, ésto no es posible corregirlo desde el código del fichero `.csd`, ya que está implícito al mecanismo de *QuteCsound*.

En lo referente al detector MIDI de nota-ON, podemos destacar que funciona correctamente, ya que al pulsar una tecla del controlador se enciende el *display* naranja correspondiente, apagándose en caso contrario. Además, se destaca la fluidez con la que se realiza ésta acción.

El sensor de CLIP también responde de forma adecuada. Para probarlo, lo que hacemos es subir el potenciómetro de ganancia (MASTER) hasta un valor cercano al máximo y pulsar diversas notas simultáneamente.

De éste modo, la amplitud de la señal final distorsionaría (`gabus>=1`), encendiéndose así el LED rojo. En el momento en que la amplitud se reduce y deja de distorsionar (`gabus<1`), el LED de CLIP se apagaría.

Dicho esto, finalizamos éste apartado mostrando el estado de los tres *displays* en modo encendido y apagado:

Figura 4.61: Detectores con los *displays* en modo activo.

Figura 4.62: Detectores con los *displays* en modo inactivo.

4.3.4.5. Selector de presets

Tras comprobar que las nuevas características implementadas en la versión definitiva del sintetizador satisfacen los propósitos iniciales, solo queda interactuar con ésta última mejora: el selector de *presets*.

El selector de *presets* se gestiona fuera del código fuente, esto es, a través de la propia interfaz del sintetizador (panel de *widgets*), manejando distintos tipos de canales reservados. Éstos conceptos ya los estudiamos en apartados anteriores.

A continuación, crearemos distintos tipos de *presets* y los almacenaremos según se explicó en el apartado 4.1.5. Inicialmente, hemos propuesto los siguientes:

1. INICIALIZACIÓN: Contiene todos los parámetros inicializados a cero para posibilitar el diseño sonoro desde el principio.

2. LEAD 1: Sintetizador sin modulación FM, con forma de onda de portadora sinusoidal y algunos efectos aplicados.

3. LEAD 2: Sintetizador sin modulación FM, con forma de onda de portadora cuadrada y algunos efectos aplicados.

4. CAMPANA TUBULAR: Imitación de una campana tubular aplicando síntesis FM con $I = 10$, $c = 5$ y $m = 7$, así como algunas variaciones en las envolventes ADSR.

5. BAJO 1: Bajo eléctrico sin modulación FM. Envolvente de portadora de tipo unidad y factor de portadora 0,5.

6. BAJO 2: Bajo eléctrico con distorsión a partir de modulación FM con $I = 20$, $c = 4,8$ y $m = 0,6$. Tiempo de relajación de moduladora de 1 s y decaimiento de portadora de 0,355 s.

7. STRING: Conjunto de cuerdas electrónicas mediante síntesis FM con $I = 1$, $c = 1$ y $m = 2$. Modificación de las envolventes ADSR para aportar mayor dinámica al instrumento.

8. KALIMBA: Imitación del instrumento *kalimba* haciendo uso de síntesis FM con $I = 2,2$, $c = 2$ y $m = 9$. Envolventes ADSR con parámetros muy bajos.

9. BAJO + PICK: Imitación de bajo eléctrico con el *pick* característico de la púa al frotar la cuerda. Parámetros de FM: $I = 8,2$, $c = 0,2$ y $m = 0,1$.

Tras haber definido los distintos *presets*, nos desplazamos por el selector (*spinbox*) y comprobamos que, efectivamente, podemos cambiar entre los *presets* creados. En la figura 4.63 puede observarse la carga satisfactoria del *preset* 4 (campana tubular).

Invitamos al lector a que explore los distintos *presets* implementados y escuche los resultados que se obtienen con éstos. Asimismo, también podrá crear nuevos o sustituir los ya existentes.

4.4. Problemas encontrados y soluciones adoptadas

A lo largo del desarrollo del presente proyecto, nos hemos encontrado con diversos *handicaps* que, afortunadamente, han sido de fácil solución.

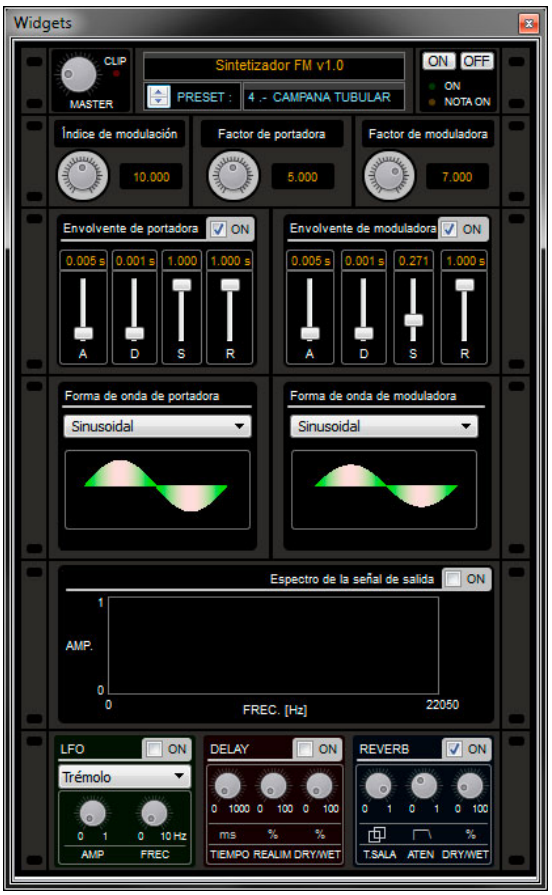


Figura 4.63: Carga satisfactoria del *preset* 4 (campana tubular).

En éste apartado se discutirán de forma breve, de modo que si el lector pudiera llegar a encontrarse con alguno de los siguientes, dispondrá de una posible guía para solventarlos en la medida de lo posible.

Uno de los principales problemas que surgieron trabajando con *QuteCsound* 0.6.0 en *Windows*, fue el hecho de que los *opcodes* `invalue` y `outvalue` consumen muchos recursos del procesador.

Si nuestro sintetizador, además, contiene un número de *widgets* elevado, aún necesitaremos mayor capacidad de cálculo, lo cual no es favorable si estamos desarrollando en un ordenador con prestaciones ajustadas. El modo de detectar si nos hemos "pasado" con el número de llamadas a dichos *opcodes* es sencillo: cuando ejecutemos el sintetizador y pulsemos algunas notas, el sonido contendrá numerosos chasquidos.

Un modo de reducir ésta carga computacional es hacer uso del *opco-*

de `metro`. Recordemos que éste *opcode* únicamente generaba una señal de reloj o metrónomo de la frecuencia especificada `kfreq` (en Hz). De éste modo, la señal valdrá 1 cada `1/kfreq` segundos.

Si ahora implementamos un condicional simple (`if`), diciendo como condición que cuando el reloj valga `1` que se envíe o se lea información de algún *widget* (mediante `outvalue` o `invalue`), lo que estaremos haciendo será ahorrar instancias a éstos *opcodes* de lectura y escritura, ya que no todos los *widgets* tienen porqué estar continuamente leyendo o enviando información.

Al ahorrar instancias a los *opcodes* `outvalue` e `invalue`, estaremos también reduciendo la carga de cómputo. Dependiendo del propósito del *widget*, tendremos que aumentar o reducir la frecuencia con la que se lee de él o se le envía información.

Por ejemplo, representar el espectro en tiempo real debe realizarse a mayor frecuencia que la comprobación de una casilla para activar la envolvente.

En cualquier caso, dicha frecuencia se debe ajustar de forma experimental hasta conseguir buenos resultados con cada *widget* en particular.

Aumentando el valor de la variable global reservada `ksmps` también se puede reducir carga computacional, aunque no es recomendable ya que también empeoramos la calidad de la señal de salida. Un consejo es ajustar `ksmps` lo más bajo posible, de modo que lleguemos a un equilibrio entre una señal con suficiente calidad y que se encuentre libre de chasquidos.

Otro problema con el que nos topamos fue que después de haber implementado los selectores de forma de onda y el analizador de espectro, las tablas de onda y el espectro no se representaban. Para evitar ésto, debemos habilitar el flag `--displays`, colocando ésta sentencia entre las etiquetas `<CsOptions>` y `</CsOptions>`.

Con respecto al generador de envolventes, primero comenzamos utilizando el *opcode* de envolventes para tiempo real `madsr`. No obstante, en vista de que no se producían resultados favorables, optamos por el uso de `linsegr`.

Éste *opcode* dio respuestas muy buenas desde un mismo principio. Únicamente, debemos tener en cuenta que como recibe parámetros de tipo nota (`i`), para introducirle variables de tipo `k` (las obtenidas por los *widgets*), tendremos que hacer *casting*, es decir, modificar su tipo de forma

momentánea, haciendo uso de la sentencia: `i(kmivariable)`.

Por último, aunque existen versiones superiores de *QuteCsound* (como la 0.7.0), se recomienda el uso de éste sintetizador en la versión en la que ha sido desarrollado (0.6.0). Probamos su ejecución en la 0.7.0, pero el espectro de la señal de salida responde de forma más lenta.

Para finalizar el capítulo, hemos creído conveniente poner a disposición del lector el siguiente vídeo tutorial sobre el sintetizador tratado, en el que se muestran algunos ejemplos de uso: `http://youtu.be/yOISuNDznUI`

Capítulo 5

Conclusiones finales

Csound constituye una potente herramienta como lenguaje de síntesis, puesto que simplifica de forma considerable la labor de programación a través de su extensa librería de *opcodes*.

Por otro lado, la interacción en tiempo real con *Csound* se ha podido llevar a cabo gracias al entorno de desarrollo *QuteCsound*. El hecho de disponer de elementos gráficos (*widgets*) que puedan establecer relaciones con el código implementado, constituye un gran recurso que debemos tener presente.

Si bien la técnica de síntesis por modulación en frecuencia ha sido desarrollada hace algunas décadas, no debemos olvidar que, por una parte, ha constituido el inicio de la era digital de la síntesis; y por la otra, que se trata una técnica cuya principal ventaja es la eficiencia. Es una técnica muy eficiente. Y es que éste aspecto en el campo de la ingeniería resulta crucial. Por consiguiente, puede resultar apropiada para sistemas cuya capacidad de cálculo sea limitada.

Aunque pueda parecer que la síntesis FM engloba una matemática algo confusa, en sí misma es una técnica simple, pues podemos generar un espectro muy rico en parciales con un número reducido de osciladores, gobernando a su vez muy pocos parámetros de control.

Por otra parte, la gran desventaja de la síntesis FM radica en que dicho control es menos intuitivo y más sensible que en las técnicas de síntesis lineal. A la par, la síntesis FM resulta poco adecuada si deseamos realizar síntesis imitativa.

El principal objetivo del proyecto tratado ha sido crear un sintetizador FM simple que forme parte del material docente del profesor para

poder enseñar, tanto a alumnos de la asignatura de *Síntesis Digital de Sonido*, como personas interesadas en éste campo, la técnica de síntesis FM, observando la repercusión que tienen los parámetros en el sonido generado, así como en su espectro.

El cumplimiento del anterior objetivo se tendrá que evaluar durante los próximos años, en los que el profesor podrá considerar si éste material contribuye a la docencia. Esperemos, de todo corazón, que así sea.

Con la lectura de éste proyecto, hemos tenido la oportunidad de aprender no solo la técnica de síntesis FM simple, sino también algunos algoritmos de FM compuesta, que constituyen las bases de los sintetizadores FM más potentes y actuales del mercado. Por otra parte, hemos abarcado los fundamentos básicos sobre el lenguaje *Csound*, así como su aplicación a la hora del desarrollo de sintetizadores que trabajen en tiempo real.

El lector es libre de utilizar, modificar o distribuir el código del presente proyecto para adaptarlo a fines didácticos y/o personales.

Dicho ésto, podemos concluir que los objetivos principales del proyecto han sido cumplidos.

5.1. Líneas de posibles trabajos futuros

Debido a la naturaleza del presente proyecto, se pueden presentar posibles líneas de trabajos futuros. A continuación, se propondrán algunas ideas tanto de proyectos de fin de carrera como de nuevas líneas de investigación.

- Ampliación y optimización del presente sintetizador (versión 2.0).

  Puede ser interesante añadir nuevos módulos y funciones al presente sintetizador, así como modificar u optimizar los ya presentes, según las necesidades del profesor para facilitar la docencia de la síntesis FM.

- Creación de un sintetizador basado en síntesis FM compuesta.

  El presente proyecto se ha centrado en síntesis FM simple. No obstante, el verdadero potencial tímbrico se encuentra en el diseño de algoritmos de la FM compuesta.

  Con lo cual, se podría implementar un sintetizador que le preguntara al usuario sobre un número de operadores, con la posibilidad de interconectarlos en diversos algoritmos (serie, paralelo, cascada,

realimentación, etc). Otra opción, podría ser crear un banco preestablecido de algoritmos, de modo que el usuario pueda seleccionar el que desee.

- Portabilidad de *Csound* al sistema *Android*.

  Constituiría una potente línea de investigación debido al creciente desarrollo que está teniendo el sistema operativo de la compañía *Google*. Dado que las capacidades de los *smartphones* y *tablets* también se encuentran en continua expansión, puede ser interesante la implementación de un sintetizador que pudiera ser controlado por un teclado inalámbrico vía *bluetooth*.

  Cabe destacar que ya existe un reproductor de archivos CSD disponible para *Android*: `http://www.csounds.com/journal/issue17/android_csd_player.html`

- Conversión de un sintetizador de *QuteCsound* a una aplicación en C++ multiplataforma.

  Si deseamos crear un sistema verdaderamente estable y eficiente que se ejecute de forma independiente, podemos pensar en realizar un *port* desde *QuteCsound* a un entorno de desarrollo de C++.

  Se propone *Qt* por ser un entorno de desarrollo multiplataforma, que además, incluye *Qt Designer*. Se trata de una extensión de dicho IDE que permite elaborar la interfaz gráfica de la aplicación (GUI) de una forma cómoda. Presenta la mayoría de *widgets* de *QuteCsound* e incluye muchas características adicionales.

  Conociendo la implementación en *Csound*, resulta más fácil el desarrollo en C++ que si partiésemos desde cero.

- Creación de un *plugin* VST basado en *Csound*.

  Finalmente, también puede considerarse la opción del diseño de un prototipo en *Csound* (sintetizador o módulo de efectos) previo paso a la construcción del VST correspondiente, a fin de que pueda ser ejecutado en cualquier *software* de producción o secuenciación musical, como *Ableton*, *Cubase*...etc.

Apéndice A

Código fuente del proyecto

A continuación, se presenta el código fuente del proyecto tratado en la presente memoria. Se trata de una posible solución al problema planteado.

```
;-------------------------------------------------------------------------
;                          SINTETIZADOR FM v.1.0
;-------------------------------------------------------------------------
; Código: Implementación de un sintetizador FM básico formado por
;         2 operadores y pensado para el uso en tiempo real.
;-------------------------------------------------------------------------
; Autor: Kristian Alonso Stenberg.
;-------------------------------------------------------------------------
; Fecha: 28 de Noviembre de 2012.
;-------------------------------------------------------------------------

;Diseño del sintetizador FM.
<CsoundSynthesizer>

;Opciones de Csound.
<CsOptions>
;Activamos este flag para poder visualizar los gráficos del interfaz.
--displays
</CsOptions>

;Diseño de la orquesta.
<CsInstruments>

;Declaramos las variables globales.
sr = 44100 ;Frecuencia de muestreo.
ksmps = 128;Número de muestras calculadas por periodo de control.
nchnls = 2 ;Estéreo.
0dbfs = 1 ;Amplitud de referencia.

;Variable global para saber si estamos en la primera iteración.
giaux init 0
;Variable global para implementar el bus de audio principal.
gabus init 0.0
```

```
;Sintetizador FM a partir de dos osciladores de interpolación cúbica con
;posibilidad de LFO para trémolo o vibrato.
instr 1
    ;Amplitud de la nota MIDI normalizada.
    iamp ampmidi 0dbfs*0.3
    ;Frecuencia de la nota MIDI.
    ifrec cpsmidi

    ;Señal de metrónomo de amplitud = 1 (cuando la fase es 0 o 1) y
    ;frecuencia 10 Hz.
    ktrig metro 10

    ;Condicional para saber si la señal de metrónomo vale 1.
    ;Con esto, ahorramos instancias a los opcodes.
    if (ktrig==1) then
        ;Leemos los valores de entrada de diversos widgets.
        kptabla invalue "menufport"  ;Forma de onda de portadora.
        kmtabla invalue "menufmod"   ;Forma de onda de moduladora.
        klfo_on invalue "lfo_on"  ;Casilla de activación del LFO.
        kmenulfo invalue "menulfo"  ;Menú desplegable del LFO.
        klfoamp invalue "lfo_amp"  ;Amplitud del LFO.
        klfofrec invalue "lfo_frec"  ;Frecuencia del LFO.
    endif

    ;Leemos el estado de los interruptores ON de las envolventes.
    konmod invalue "onmod"
    konport invalue "onport"

    ;Almacenamos los parámetros de índice de modulación y factores de
    ;frecuencia de portadora y moduladora.
    kindex invalue "indexknob"
    knmod invalue "nmknob"
    knport invalue "ncknob"

    ;Calculamos la frecuencia y amplitud de la señal moduladora.
    kfmod = ifrec*knmod
    kamod = kindex*kfmod

    ;Envolvente de la señal moduladora.
    if (konmod==1) then ;Si se activa la casilla ON.

        ;Leemos los parámetros ADSR de los faders.
        katmod invalue "atmod" ;Tiempo de ataque (s).
        kdmod invalue "dmod"   ;Tiempo de decaimiento (s).
        ksmod invalue "smod"   ;Amplitud de sostenimiento.
        krmod invalue "rmod"   ;Tiempo de relajación (s).

        ;Generamos la envolvente correspondiente de la señal moduladora.
        amenv linsegr 0, i(katmod), 1, i(kdmod), i(ksmod), i(krmod), 0

    else ;Si se desactiva la casilla ON.
        amenv linsegr 0,0.001,1,0.001,0 ;Envolvente de amplitud unidad.
    endif
```

```
   ;Envolvente de la señal portadora.
    if (konport==1) then ;Casilla ON activada.

        ;Obtenemos los parámetros ADSR de los faders.
        kaport invalue "aport" ;Tiempo de ataque (s).
        kdport invalue "dport" ;Tiempo de decaimiento (s).
        ksport invalue "sport" ;Amplitud de sostenimiento.
        krport invalue "rport" ;Tiempo de relajación (s).

        ;Generamos la envolvente correspondiente de la señal portadora.
        apenv linsegr 0, i(kaport), 1, i(kdport), i(ksport), i(krport), 0

    else ;Casilla ON desactivada.
        apenv linsegr 0,0.001,1,0.001,0 ;Envolvente de amplitud unidad.
    endif

    ;Señal moduladora.
    amod oscil3 kamod*amenv, kfmod, i(kmtabla+1)

    ;Creamos un LFO para generar un trémolo o vibrato si así se desea.
    kenvlfo linenr klfoamp, 0.01, 0.01, 0.001 ;Envolvente del LFO.
    klfo lfo kenvlfo, klfofrec, 0 ;Implementamos el LFO.

    ;Señal portadora.
    if (klfo_on==0) then ;Casilla ON del LFO desactivada.

        ;Señal portadora sin LFO aplicado.
        aport oscil3 iamp*apenv, ifrec*knport+amod, i(kptabla+1)

    elseif (klfo_on==1&&kmenulfo==0) then ;LFO ON y trémolo activado.

        ;Señal portadora con LFO aplicado (Trémolo).
        aport oscil3 iamp*(apenv+a(klfo)), (ifrec*knport)+amod, i(kptabla+1)

    elseif (klfo_on==1&&kmenulfo==1) then ;LFO ON y vibrato activado.

        ;Señal portadora con LFO aplicado (Vibrato).
        aport oscil3 iamp*apenv, (ifrec*knport+amod)*(1+a(klfo)), i(kptabla+1)

    endif

    ;Mandamos la señal de salida (portadora) al bus de audio principal.
    gabus = aport + gabus

endin


;Módulo para efecto "Delay".
instr 100
    ;Almacenamos el valor del interruptor ON del módulo DELAY.
    kdelayon invalue "delay_on"

    ;Si el interruptor está en ON, generamos el "Delay".
    if(kdelayon==1) then
```

```
        ;Señal de metrónomo con f=10 Hz para ahorrar llamadas a los opcodes
        ;"invalue".
        ktrig metro 10

        ;Si el metrónomo vale 1, leemos los parámetros del "Delay".
        if(ktrig==1) then
            kdrywet invalue "delay_dw" ;Cantidad de efecto (%).
            ktiempo invalue "delay_t"  ;Tiempo de retardo (s).
            krealim invalue "delay_r" ;Realimentación (%).
        endif

        adelay delayr 1 ;Generamos una línea de retardo de 1 s.
        awet deltapi ktiempo ;Creamos la señal retardada "ktiempo" segundos.
        delayw gabus + (awet*krealim) ;Añadimos a la línea de retardo la
                                      ;señal de audio del bus principal.

        ;Añadimos al bus de audio principal la mezcla de señal "seca" y
        retardada.
        gabus = (1-kdrywet)*gabus + kdrywet*awet
    endif
endin

;Módulo para efecto "Reverb"
instr 101

    ;Obtenemos el valor de la casilla ON del módulo REVERB.
    krev_on invalue "rev_on"

    ;Si la casilla ON se encuentra activada, se implementará el "Reverb".
    if(krev_on==1) then

        ;Señal de metrónomo con f=10 Hz para optimizar las llamadas a los
        ;opcodes "invalue".
        ktrig metro 10

        ;Si la amplitud de la señal de metrónomos es igual a la unidad.
        if(ktrig==1) then
            ;Leemos el estado de los knobs del módulo de reverberación.
            ktam invalue "rev_sala"  ;Factor del tamaño de la sala.
            kataf invalue "rev_ataf" ;Factor de atenuación de altas
                                     frecuencias.
            kdrywet invalue "rev_dw" ;Cantidad de efecto (%).
        endif

        ;Implementamos la señal reverberada.
        awet, awet freeverb gabus, gabus, ktam, kataf

        ;Introducimos la mezcla de señales "seca" y reverberada en el bus
        ;principal de audio.
        gabus = (1-kdrywet)*gabus + kdrywet*awet
    endif
endin
```

```
;Instrumento que almacena diversos detectores, representa el espectro en
;tiempo real y saca la señal de salida amplificada previamente.
instr 102

    ;Guardamos el estado del botón OFF.
    koff invalue "_Stop"

    ;Si estamos en la primera iteración y no se ha pulsado el botón OFF.
    if (giaux==0&&koff==0) then
        ;Encendemos el LED ON.
        outvalue "ondisp", "·"
        ;Ponemos a 1 "giaux" para saber que ya no estamos en la primera
        iteración.
        giaux = 1

        ;Si se pulsa el botón OFF.
    elseif (koff==1) then
        ;Apagamos los LED's ON, NOTA ON y CLIP.
        outvalue "ondisp", " "
        outvalue "clip", " "
        outvalue "notaon", " "
        ;Ponemos a cero el bus de audio principal.
        gabus = 0
    endif

    ;Obtenemos información de la nota MIDI.
    kstatus, kchan, kdata1, kdata2 midiin

    ;Detector de nota ON.
    if (kstatus==144) then ;Si se ha recibido NOTA ON.
        ;Encendemos el LED NOTA ON.
        outvalue "notaon", "·"
    elseif (kstatus==128||kdata2==0) then ;Si se ha recibido NOTA OFF.
        ;Apagamos el LED NOTA ON.
        outvalue "notaon", " "
    endif

    ;Obtenemos el nivel de la señal de salida del knob del módulo MASTER.
    knivel invalue "nivel"

    ;Multiplicamos el nivel leído por la señal de bus de audio principal
    ;para obtener la señal de salida.
    asalida=gabus*knivel


    ;Detector de distorsión (clipping).
    ktrigclip metro 10 ;Señal de metrónomo de f=10 Hz.
    kclip max_k asalida, ktrigclip, 1 ;Obtenemos el valor máximo en 0.1
                                      ;segundos de la señal de salida.
    ;Si el valor máximo de la señal de salida distorsiona.
    if (kclip>=1) then
        ;Encendemos el LED CLIP.
        outvalue "clip", "·"
```

```
    ;Si no distorsiona.
    else
        ;Apagamos el LED CLIP.
        outvalue "clip", " "
    endif

    ;Sacamos por el dispositivo de audio la señal de salida.
    outs asalida, asalida

    ;Obtenemos el valor de la casilla ON del módulo del
    ;espectro de la señal de salida.
    kespectro invalue "espectro_on"

    ;Si se ha activado la casilla ON.
    if (kespectro==1) then
        ;Representamos el espectro de la señal.
        dispfft asalida, 0.01, 256
        outvalue "espectro", 5
    endif

    ;Ponemos a cero el bus de audio principal para que no se produzca
    ;realimentación positiva en el sonido.
    gabus = 0
endin

</CsInstruments>

;Diseño de la partitura.
<CsScore>
;Definimos las tablas necesarias.

;Ciclo de onda sinusoidal.
f 1 0 4096 10 1
;Ciclo de onda cuadrada con 8 armónicos.
f 2 0 4096 10 1 0 [1/3] 0 [1/5] 0 [1/7] 0 [1/9] 0 [1/11] 0 [1/13] 0 [1/15]
;Ciclo de onda triangular con 8 armónicos.
f 3 0 4096 10 1 0 -[1/9] 0 [1/25] 0 -[1/49] 0 [1/81] 0 -[1/121] 0 [1/169] 0 -[1/225]
;Ciclo de onda diente de sierra con 8 armónicos.
f 4 0 4096 10 1 [1/2] [1/3] [1/4] [1/5] [1/6] [1/7] [1/8]
;Ciclo de onda pulso con 8 armónicos.
f 5 0 4096 10 1 1 1 1 1 1 1 1

;Llamamos a los instrumentos.
i 100 0 3600 ;Efecto "Delay".
i 101 0 3600 ;Efecto "Reverb".
i 102 0 3600 ;Detectores, espectro y señal de salida.
e
</CsScore>
</CsoundSynthesizer>
```

Bibliografía

[1] J.M. Chowning, "*The Synthesis Of Complex Audio Spectra By Means of Frequency Modulation*", Journal of Audio Engineering Society, Vol. 21, Nº7, 1973.

[2] R. Boulanger, "*The Csound Book: Perspectives in Software Synthesis, Sound Design, Signal Processing and Programming*", MIT Press, 2000.

[3] R.T. Dean, "*The Oxford Handbook of Computer Music*", Oxford University Press, 2009.

[4] P. Manning, "*Electronic and Computer Music*", Oxford University Press, 2013.

[5] C. Dodge & T.A. Pierce, "*Computer Music: Synthesis, Composition and Performance*", Schrimer Books, 2nd Ed., 1997.

[6] E.R. Miranda, "*Computer Sound Design: Synthesis Techniques and Programming*", Focal Press, 2nd Ed., 2002.

[7] F. Bowman, "*Introduction to Bessel Functions*", Courier Dover Publications, 1958.

[8] B. Vercoe, "*The Canonical Csound Reference Manual: Version* 5.19.0", MIT Media Lab, 2013.

[9] J. Langmead, "*A Beginner's Guide to Volume Envelopes in Csound*", Csound Journal, Issue 11, 2009.

[10] J. Hearon, "*Tour of Oscillators*", Csound Journal, Issue 3, 2006.

[11] J.M. Iñesta, "*Apuntes sobre Síntesis Digital de Sonido*", Universidad de Alicante.

[12] J.M. Sáez, "*Apuntes sobre Fundamentos de Programación*", Universidad de Alicante.

[13] A. Cabrera, "*QuteCsound, a Csound Frontend*", Sonic Arts Research Centre, Queen's University Belfast, 2010.