UNIVERSIDAD SAN FRANCISCO DE QUITO

Implementación de tratamiento digital de la dinámica de señales de audio en C++ utilizando especificación VST

José Luis Pinos Vargas

Tesis de grado presentada como requisito para la obtención del título de Ingeniero Eléctrico / Electrónico

Quito

Diciembre del 2010

Universidad San Francisco de Quito

Politécnico

HOJA DE APROBACION DE TESIS

Implementación de tratamiento digital de la dinámica de señales de audio en C++ utilizando especificación VST

José Luis Pinos Vargas

Daniel Cárdenas, Ph. D.
Director de Tesis
Miembro del Comité de Tesis _____

René Játiva, DEA.
Miembro del Comité de Tesis _____

Santiago Navarro, Ph. D.
Miembro del Comité de Tesis _____

Fernando Romo, M. Sc.
Decano del Colegio de Ciencias e Ingeniería _____

Quito

Diciembre del 2010

Agradezco a mis padres, hermanos y amigos por su apoyo incondicional a largo de este accidentado pero satisfactorio camino. A Christine por su infinita paciencia y comprensión, la cual espero algún día poder retribuir.

"...pero seguí pa lante y pa lante
has como yo nunca eché pa tras
ni pa coger impulso"

HECTOR LAVOE, *Todo tiene su final*

Resumen

Mientras el poder de procesamiento de los sistemas informáticos personales aumenta año tras año, la cantidad de procesamiento de señales de audio en tiempo real que se realizan en tales sistemas también aumenta. Lo que se solía hacer gracias a unidades externas de efectos, ahora se puede realizar dentro del sistema de computación utilizando módulos de código de procesamiento de señal, los plug-ins. Esta tesis describe el desarrollo de una serie de plug-ins VST. Primeramente se programó un prototipo en MATLAB, y luego se realizó una implementación en C++ utilizando el Kit de Desarrollo de Software VST (SDK VST). El código fuente C++ se compiló en Mac, resultando en versiones del plug-in que pueden ser utilizadas en cualquier aplicación anfitriona VST. Escribir o desarrollar un plug-in libera al programador de la carga de lidiar directamente con detalles de la interfaz de audio y especificaciones de la interfaz gráfica de usuario (GUI), ya que la aplicación anfitriona se encarga de esto. Así pues, esta es una forma interesante para desarrollar algoritmos de audio de procesamiento digital de señal, ya que la aplicación anfitriona también brinda la oportunidad de escuchar y medir el desempeño del algoritmo del plug-in implementado.

Abstract

The processing power of computer systems grows every year. In this context, signal processing performed by these systems grows as well. Thanks to plug-ins, computer systems can now process audio signals using just code modules. These tasks used to be done by racks of external effects. This thesis describes the development of four VST plug-ins. First, a prototype was programmed using MATLAB, and then a C++ implementation was made using the VST Software Development Kit. The C++ source code was compiled in Mac, resulting in different plug-in versions that can be used in any VST host application. Writing a plug-in makes the programmer's job a little easier since he doesn't have to deal directly with audio interfaces or graphic user interfaces, because the host application does that job. It is thus an interesting alternative to start developing audio processing algorithms, since the host application provides the opportunity to hear and measure plug-in's performance.

TABLA DE CONTENIDOS

Capítulo 1

1.1 Introducción

Desde los inicios de la reproducción y transmisión de señales de audio ha sido importante poder controlar automáticamente la amplitud de la señal para compensar las limitaciones impuestas por los formatos de reproducción y transmisión. Tomemos por ejemplo el vinilo: al contar con un determinado espacio físico para grabar una señal, la amplitud debía ser controlada para que la integralidad de la señal tuviera espacio de ser grabada. En la actualidad las limitaciones son impuestas por similares razones, solo que la amplitud depende del número de bits con los que una señal es codificada. En el caso de las transmisiones, los organismos estatales que las controlan imponen valores máximos de amplitud a los cuales se difunden los programas de radio y televisión, con la finalidad de homogenizar el nivel o volumen de escucha entre cada estación o canal.

Este tratamiento es realizado automáticamente y no puede hacerse de manera manual en primer lugar porque requeriría que una persona se encargue permanentemente de ajustar el nivel y en segundo lugar porque el oído humano es incapaz de oír ciertos transitorios que tienen un nivel muy alto o muy bajo. El procesamiento dinámico es un procesamiento automático que maneja la dinámica de una señal. Por dinámica de una señal entendemos la diferencia de amplitud, en decibeles, entre el nivel más bajo y el nivel más alto de una señal.

A pesar de que originalmente el procesamiento dinámico de las señales respondía a necesidades técnicas, pronto su uso se popularizó tanto que se convirtió una herramienta artística. Los tratamientos dinámicos se usan con mucha frecuencia en producciones musicales, de hecho en muchas consolas cada canal cuenta con una sección entera de distintos tipos de tratamientos dinámicos.

Las computadoras juegan un papel muy importante en la producción musical en la actualidad. Sistemas de computación, con aplicaciones adecuadas de hardware y software, son utilizados para grabar, editar, mezclar, masterizar y difundir por Internet. La utilización de varios tipos de procesamiento de audio como ecualización, delay, reverberación y compresión dinámica los cuales solían ser realizados por hardware, análogo o digital, externo es ahora realizado, en su mayoría, dentro del mismo sistema de computación. En tales sistemas, el procesamiento de señal no se realiza por un chip DSP especializado, sino por el CPU de ese mismo sistema. Ejemplos típicos de computadoras personales que utilizan sistemas operativos comerciales son Motorola, IBM e Intel para MacOS y AMD e Intel para Windows. Las partes de procesamiento de señal de aplicaciones de software de audio normalmente se ubican en módulos de código separados – llamados plug-ins, los cuales se cargan a la aplicación básica para extender su funcionalidad.

1.2 Enfoque y limitaciones de la tesis

El trabajo realizado durante el desarrollo de esta tesis se enfoca en la investigación de algoritmos para tratamiento de la dinámica de la señales de audio y luego la implementación de estos algoritmos en forma de plug-ins. Se trata la teoría básica de procesamiento digital aplicada al contexto de programación informática dirigida al sonido, así como el proceso de desarrollo de un plug-in de audio que sea compatible con varias aplicaciones que corren actualmente en MacOS. También se examina las partes de la estructura interna de un plug-in de audio VST escrito en C++. El desarrollo de plug-ins VST se subdivide en dos partes – el desarrollo de un prototipo en un entorno de desarrollo informático como es MATLAB, y la implementación final del plugin en C++. El tema se concentra más en implementar varios algoritmos

específicos de efectos de audio – plug-ins para tratamiento dinámico. Algunas ecuaciones matemáticas se nombran en la tesis para explicar el algoritmo DSP implementado, pero el trabajo busca presentar conceptos y no hacer demostraciones matemáticas.

El diseño de un prototipo analógico de este tipo de tratamiento es también parte del trabajo de esta tesis, sin embargo, por varias razones que serán mencionadas posteriormente, se ha decido enfocarse únicamente en la aplicación digital de este tipo de procesamiento. Aun así, el autor cree conveniente hacer una breve reseña del tratamiento analógico y mostrar el trabajo realizado en esta área.

1.3 Organización de la tesis

En el capítulo dos se hace un resumen sobre el origen de estos tratamientos en el dominio analógico ya que su arquitectura es la misma que usan en la actualidad este tipo de procesamientos en el dominio digital. De igual manera se explica rápidamente la tentativa de realizar un prototipo analógico. El capítulo tres hace una revisión de la teoría fundamental de DSP con énfasis en el procesamiento dinámico de señales de audio, así como la explicación de los tratamientos que son objeto de esta tesis. En el capítulo cuatro se examina detalladamente el desarrollo y resultados del prototipo en MATLAB. Una descripción general de los plug-ins de audio y los detalles de la implementación son descritos en el capítulo cinco. Los resultados de probar el plug-in compilado en varias aplicaciones anfitrionas en MacOS se presenta en el sexto capítulo. Se realizó un análisis de la respuesta en del plug-in, y se muestran las diferencias en las interfaces graficas de usuario por defecto. El último capítulo contiene una conclusión y también sugerencias para futuras mejoras que podrían realizarse al plug-in.

1.4 Motivación

Como ingeniero de sonido, la principal motivación de este proyecto es familiarizarme con todos los principios que se encuentran detrás del funcionamiento de una de mis principales herramientas de trabajo. Es muy importante obtener los conocimientos para ser capaz, en un futuro, de diseñar herramientas innovadoras para el tratamiento de señales de audio.

La ingeniería de sonido es una profesión un poco desvalorizada. De hecho, su solo nombre es un poco genérico ya que abarca un sin número de áreas donde puede emplearse. Como consecuencia, con mucha frecuencia se piensa, erróneamente, que cualquier persona con un poco de conocimiento sobre el tema puede realizar este trabajo. Un ingeniero de sonido es ante todo una persona que conoce muy bien cómo funcionan las herramientas que usa. Las señales de audio se transmiten y se tratan eléctricamente, por lo que es determinante tener conocimientos sobre los dispositivos electrónicos (en el caso del tratamiento analógico) y teorías detrás del procesamiento digital de señales.

En el contexto del desarrollo artístico que vive en la actualidad el Ecuador, es importante contar con personas con las capacidades ingenieriles de desarrollar herramientas al servicio de este crecimiento artístico.

La idea no es que este proyecto sea solo un requisito de graduación sino una muestra de que, tecnología y herramientas de este tipo también pueden desarrollarse en nuestro país, y es más, podemos realizarlo ingenieros locales.

En definitiva, el proyecto es de gran utilidad. Lo que se intenta aquí es realizar un diseño básico y de fácil uso.

1.5 Definición

El rango dinámico de una señal de audio se define como la relación logarítmica del máximo al mínimo de la amplitud de la señal y se mide en decibeles. El rango dinámico de una señal de audio se encuentra entre 40 y 120 decibeles. La combinación de medición de nivel y el ajuste del nivel de la señal de salida en base a esta medición se conoce como control de rango dinámico. El control de rango dinámico de las señales de audio es utilizado en varias aplicaciones para hacer coincidir el comportamiento dinámico de la señal con determinadas necesidades. Al grabar, el control de rango dinámico protege de sobrecargas a los convertidores A/D o se utiliza dentro del camino de la señal para optimizar la utilización de todo el rango dinámico del sistema de grabación. Para eliminar el ruido de los niveles bajos se utilizan compuertas de ruido para que la señal de audio pase a partir de determinados niveles. Al reproducir música en un carro, centro comercial o una discoteca, la dinámica tiene que coincidir con las características sonoras específicas al entorno. Por lo tanto, el nivel de señal se mide desde la señal de audio y una señal de control es generada, la cual a su vez afecta el nivel de salida. Este control de volumen se adapta al nivel de entrada. El procesamiento dinámico de señales de audio es realizado por aparatos de amplificación donde la ganancia esta automáticamente controlada por el nivel de la señal de entrada. Discutiremos limitadores, compresores, expansores y compuertas de ruido, por lo que es importante familiarizarse con algunos términos que se emplearán a lo largo de esta tesis, los cuales, por ser comúnmente términos en inglés, se referirá a ellos en ese idioma en ocasiones:

- Attack time o tiempo de ataque: tiempo en que la señal pasa del 10% al 90% de su valor máximo.

- Release time o tiempo de liberación: tiempo en que la señal pasa del 90% al 10% de su valor máximo.
- Hold time o tiempo de retención: se aplica en las compuertas de ruido y se refiere al tiempo que la compuerta se queda abierta dejando pasar la integralidad de la señal.
- Threshold o umbral: nivel de volumen que determina la activación o desactivación de un tratamiento dinámico.
- Ratio o relación entrada/salida: relación entre el nivel de salida y el nivel de entrada del sistema.
- dB: El decibel es usado para medir, entre otras cosas, el nivel del sonido. El dB es una unidad logarítmica usada para describir una relación. La relación puede ser de potencia, de presión del aire, de voltaje o de intensidad sonora.
- dBm: el dBm es la unidad logarítmica que describe la relación entre determinada potencia y un valor de referencia estandarizado que es 1mW.
- dBu: el dBu es la unidad logarítmica que describe la relación entre determinado voltaje y un valor de referencia estandarizado que es 0.775V. Este valor se deriva de una carga de 600Ω disipando 0dBm (1mW).
- dBFS o dB Full Scale: el dBFS representa la amplitud de una señal de audio comparada con el nivel máximo que un dispositivo digital puede codificar antes de que la señal se distorsione.

Capítulo 2 Tratamiento analógico

2.1 Consideraciones teóricas

El funcionamiento de un control de rango dinámico de las señales de audio está compuesto esencialmente por dos elementos: un detector de nivel y un amplificador con ganancia variable. La señal de salida del detector de nivel es un voltaje DC que representa la señal AC de entrada. El detector de nivel puede ser un detector de nivel pico, un detector de nivel medio o un verdadero detector RMS (Root Mean Square). La señal detectada corresponde al máximo nivel de la misma. Por otro lado, el detector de nivel medio calcula el nivel promedio de la señal y sus constantes de tiempo son similares a las del detector RMS. El verdadero detector RMS es el único detector que se relaciona directamente con la potencia de la señal, independientemente de la forma de onda de la señal. El nivel RMS de un voltaje AC es equivalente al voltaje DC que genera la misma cantidad de potencia real en una carga resistiva. Si conocemos la forma de onda de la señal, como por ejemplo un seno o una onda cuadrada, entonces el detector de picos o el detector de nivel medio, son suficientes para determinar el nivel RMS de la señal. Lamentablemente la forma de onda de una fuente musical es impredecible. Por esta razón el detector de picos y el detector de nivel medio no son los más apropiados para aplicaciones audio.

El amplificador con ganancia variable tiene normalmente tres puertos: entrada, salida y control de ganancia. El amplificador de ganancia variable puede ser un divisor de voltaje con una impedancia shunt variable (transistores JFET, opto-resistencias), un amplificador operacional de transductancia (Operational transductance amplifier, OTA), un amplificador controlado por voltaje (Voltage controlled amplifier, VCA), un atenuador controlado digitalmente, o un multiplicador en una unidad DSP.

La opción de un divisor de voltaje e impedancia shunt tiene las desventajas de tener mucha distorsión, rango dinámico limitado y una función de transferencia impredecible. El OTA tiene la ventaja de tener una función de transferencia definida y un mejor rango dinámico. Sin embargo, la distorsión en el desempeño de este tipo de componentes no es ideal para aplicaciones profesionales de audio. Un atenuador controlado digitalmente es más difícil de implementar ya que se necesita digitalizar el nivel RMS. Otro problema es que la atenuación es más difícil de implementar ya que el procesador no puede controlar entre muestras. Los VCAs son los amplificadores con ganancia variable que tienen el mejor desempeño. Los VCAs más populares tienen control exponencial de ganancia. La ventaja es que la función de transferencia exponencial es lineal en decibelios. Para poder usar estos VCAs en aplicaciones de control de dinámica de señales de audio es preferible que el detector de nivel tenga una salida logarítmica. En otras palabras, la salida DC del detector es una representación logarítmica de la señal AC de entrada [6].

2.2 Detector RMS

El nivel RMS de una señal AC *V(t)* es el la raíz cuadrada de la integral del cuadrado de la señal durante un determinado periodo de tiempo y evaluada durante el mismo periodo. La fórmula matemática corresponde a la ecuación 2.1:

$$V_{RMS} = \lim_{T\to\infty} \sqrt{{}^{1}\!/_{T} \int_{t_0}^{T+t_0} V_{in}^2(t).\,dt} \qquad \text{(Ec.2.1)}$$

donde V_{RMS} es el valor RMS de la onda de entrada $V_{in}^2(t)$ y T es el tiempo durante el cual se realiza la medición.

Un verdadero detector RMS da como resultado una señal DC que respeta la ecuación 2.1, es independiente de la forma de onda de la señal de entrada. Un pseudo detector RMS puede construirse multiplicando la salida de un detector de picos por un

factor. Sin embargo existe la limitación de que este factor varía en función de la forma de la onda, por ejemplo en el caso del coseno este factor es de raíz cuadrada de dos.

Los detectores RMS han evolucionado en el tiempo usando diferentes tecnologías. Uno de los primeros métodos para medir el nivel fue mediante la medición de los cambios de temperatura generados en un elemento resistivo al hacer pasar la señal por este elemento. Este método se sigue usando en los detectores RMS de alta frecuencia. Este tipo de detector tiene un ancho de banda muy grande sin embargo tiene un rango dinámico reducido, lo que hace que no sea óptimo para aplicaciones de audio. De hecho, el rango dinámico de este método es de 40 a 60 dB y un tiempo de ataque muy lento, lo que lo hace completamente inadecuado para tratar señales de audio. El progreso de la tecnología bipolar en los años 70 hizo posible la introducción de detectores RMS de estado sólido los cuales no usan el calor para hacer sus medidas. El rango dinámico de estos detectores es muy superior, del orden de los 80 dB, además tiempos de ataque de menos de 1 ms son posibles.

Como se mencionó anteriormente, el detector RMS puede tener una salida lineal o logarítmica. Las figuras 2.1 y 2.2 muestran respectivamente las dos configuraciones para detección lineal y logarítmica.

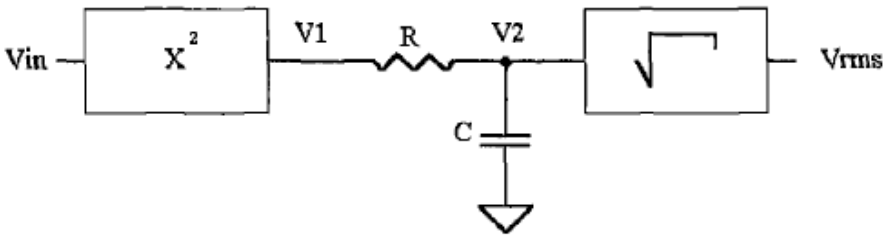


Figura 2.1. Bloques de un detector RMS con filtro en el dominio lineal [6]

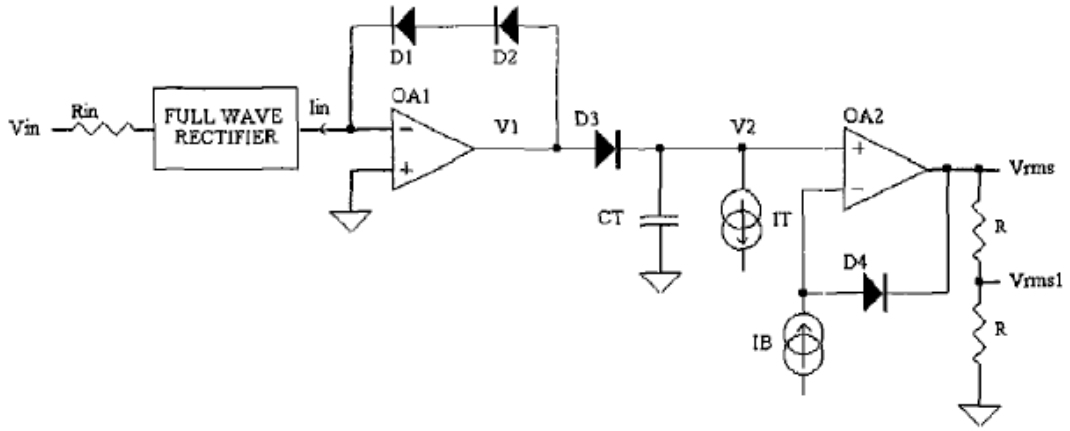


Figura 2.2. Diagrama circuital simplificado del detector RMS con filtro en dominio logarítmico [6]

En un detector lineal la señal de entrada es elevada al cuadrado, luego integrada por un tiempo finito y finalmente se extrae su raíz cuadrada. El problema de este enfoque es que la operación de elevar al cuadrado la señal necesita de un rango dinámico muy elevado. Por ejemplo, para detectar señales de audio profesionales que pueden alcanzar el valor de +24dBu con una dinámica de 80 dB, el voltaje pico de la salida del bloque de elevación al cuadrado debería oscilar entre 3.03µV y 303 V. El rango dinámico requerido por este bloque seria de 160 dB. Una manera de resolver este problema es comprimir la señal que ingresa al detector RMS para a la salida multiplicarlo por el mismo factor por el que fue escalado a la entrada. El integrador es un filtro pasabajos de primer orden. La constante de tiempo determina el rizado y la repuesta transitoria en salida del detector. El detector RMS logarítmico es más flexible en lo que respecta el voltaje requerido por el rango dinámico. La señal de entrada necesita ser rectificada ya que la función logaritmo solo está definida para valores positivos. El siguiente paso es tomar el logaritmo de la señal rectificada. Como estamos en dominio logarítmico la operación para elevar al cuadrado no es más que una

multiplicación de la señal por dos. De esta manera el rango dinámico requerido por la salida se reduce a 25.3 dB. La señal es integrada por un filtro logarítmico de primer orden. Una de las características limitantes de este detector es la corriente que necesita el integrador logarítmico. La corriente en el diodo del filtro logarítmico es proporcional al cuadrado de la corriente de entrada. Por lo que el detector RMS basado en filtros logarítmicos requiere un rango dinámico de corriente grande, en lugar de un rango dinámico de voltaje grande.

2.3 Topologías de controladores de rango dinámico

2.3.1 Controladores lineales

- Feedforward

El diagrama de bloques de un compresor lineal feedforward se muestra en la figura 2.3.

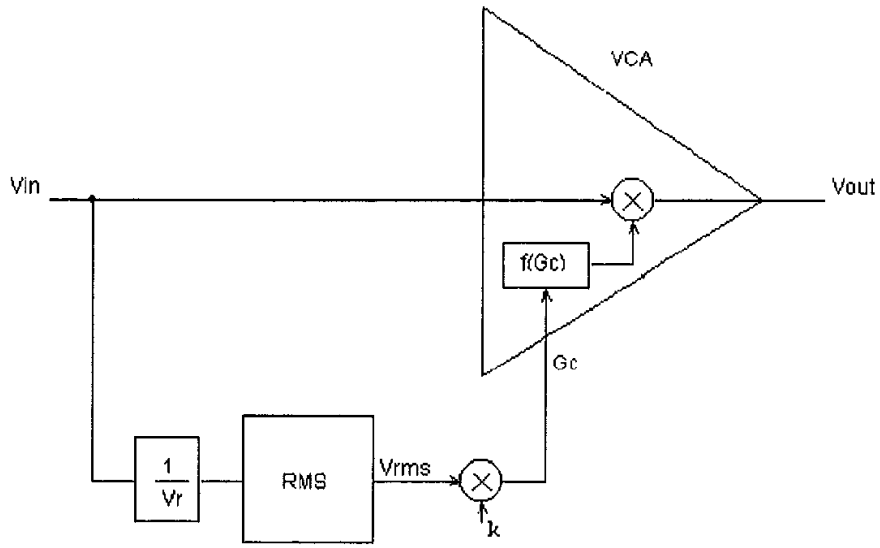


Figura 2.3. Topología feedforward de un compresor

El detector de nivel RMS y el VCA son elementos lineales. La entrada del detector RMS está escalada a un voltaje de referencia. A este nivel de referencia la

salida del detector RMS lineal es igual a uno. El voltaje en el puerto de control *Gc* se aplica a un bloque interno matemático. El voltaje de salida es el producto del voltaje de entrada por el voltaje de control de ganancia. Los voltajes de salida y entrada tienen la misma polaridad. La salida del detector RMS se aplica al puerto de control de ganancia del VCA lineal [5].

- **Feedback**

  El diagrama de bloques de un compresor lineal feedback se muestra en la figura.

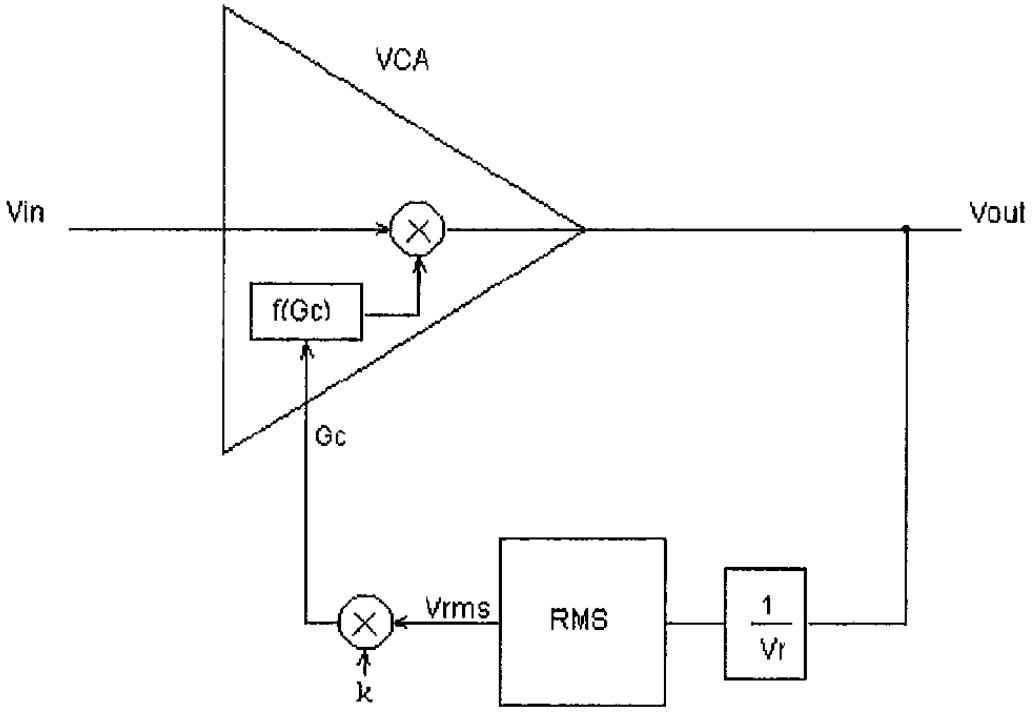


Figura 2.4. Topología feedback de un compresor

El detector RMS y el VCA son en este caso también dispositivos lineales. La entrada del detector es dividida por el nivel de referencia Vr, y su salida es de igual manera uno cuando se aplica el nivel de referencia al detector RMS.

2.3.2 Controladores logarítmicos

- **Feedforward/Feedback**

Los diagramas de bloques de compresores feedforward o feedback logarítmicos son los mismos que los diagramas correspondientes de los compresores lineales en sus dos topologías. La diferencia es que, en este caso, el detector RMS es logarítmico y el VCA es exponencial. El voltaje de entrada del detector RMS es de igual manera escalado a un nivel de referencia Vr. Cuando le aplicamos este nivel de referencia a la entrada del detector RMS su salida es cero. Se añade un bloque adicional entre la salida del detector y el puerto de control del VCA. Este bloque define el la relación de entrada salida que debe aplicarse al control de nivel.

Capítulo 3 Tratamiento digital: Bases

Esta sección provee un breve repaso de los conceptos que forman la base de los temas que serán tratados en los siguientes capítulos.

3.1 Efectos digitales de audio

En esta tesis no se da ninguna descripción de algoritmos para diferentes efectos digitales de audio. Este tema ha sido tratado por otros autores. Zölzer [1] hace un repaso completo del tema, cubriendo filtros, delays, moduladores, procesamiento dinámico y efectos espaciales, con implementaciones de software utilizando MATLAB. Los efectos digitales de audio se clasifican en: efectos de modulación de amplitud, efectos de modulación de frecuencia, efectos espaciales, filtros y compresión de dinámica.

3.2 Teoría básica DSP

El desarrollo de plug-ins de audio requiere conocimiento no solo del lenguaje de programación y del Framework que será utilizado para la implementación, sino también de teoría básica de DSP. El desarrollo de algoritmos DSP es un tema avanzado que involucra conocimientos amplios de matemáticas. Esto está fuera del alcance de esta tesis.

3.2.1 Señales y sistemas

Las señales son patrones de variaciones que representan o codifican información. Tales patrones evolucionan con el tiempo para crear formas de onda. Un ejemplo de una señal en tiempo continuo (señal analógica) es la variación de voltaje de salida de un micrófono. Esta señal puede ser matemáticamente representada por una función x de una variable continua t

$$x(t) \quad \text{(Ec 3.1)}$$

en donde *t* se refiere al tiempo. Al muestrear la señal de tiempo continuo a intervalos regulares de tiempo, se obtiene una señal de tiempo discreto (señal digital)

$$x[n] = x(nT_S) \quad \text{(Ec. 3.2)}$$

en donde *n* es número entero, T_S es el período de muestreo

$$T_S = \frac{1}{f_s}. \quad \text{(Ec.3.3)}$$

Y f_s es la frecuencia de muestreo. La señal *x[n]* es una secuencia de números indexadas por el entero n. Los números en *x[n]* corresponde a los valores muestreados de *x (t)* cada T_S segundos. En esta tesis la se utilizan los corchetes […] para denotar una señal en tiempo discreto y los paréntesis (…) para denotar una señal en tiempo continuo. El uso de corchetes para secuencias coincide con la sintaxis de arreglos de números en C++. Un arreglo llamado x que contiene cuatro muestras de audio digital tipo flotante se declara como

float x[4];

y las muestras individuales se podrían acceder como x[0], x[1], x[2] y x[3].

En un sentido muy general, un sistema sobre señales para producir nuevas señales [7]. Utilizando esta definición, la ecuación 3.2 podría ser vista como un sistema donde la entrada es una señal de tiempo continuo y la salida es una señal de tiempo discreto. El sistema es llamado un convertidor A/D (Análogo Digital).

Un sistema de tiempo discreto toma una señal de entrada *x[n]* y produce una señal de salida correspondiente *y[n]*. Esto puede ser representado visualmente por el diagrama de bloques en la figura 3.1.

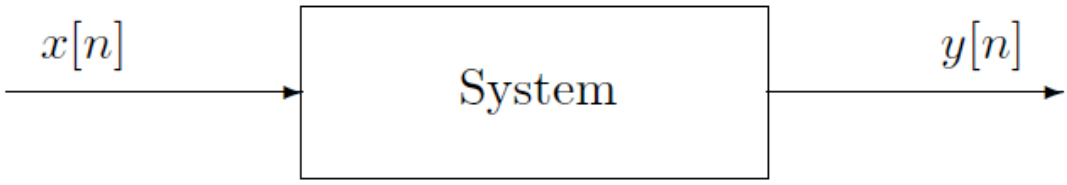


Figura 3.1. Diagrama de bloques de un sistema de tiempo discreto [11]

3.2.2 Cuantización

Como lo mencionamos anteriormente, el sistema de hardware para convertir una señal analógica en una señal digital es un convertidor A/D. Debido a los problemas del mundo real como el jitter y la cuantización de valores de muestreo con una resolución finita, el convertidor A/D es solo una aproximación del muestreo perfecto del convertidor de tiempo continuo a tiempo discreto ideal.

La resolución de la cuantización de los valores de muestreo en *x[n]* afecta la calidad de audio de la señal. Para los plug-ins VST, las muestras de audio se manejan como números de tipo punto flotante de 32 bits [3]. Los valores de muestreo son normalizados a un rango determinado que es de -1.0 a +1.0. Así un valor de muestreo de 1.0 corresponde a 0dBFS (decibeles Full Scale), un valor de 0.5 corresponde a -6 dBFS, etc.

La utilización de valores de muestreo de tipo punto flotante de 32 bits en el rango de -1.0 a +1.0 significa que la resolución interna de un plug-in VST es mayor a una resolución de 16 bits tipo enteros (estándar de CD de audio). También da suficiente espacio para lidiar con una sobrecarga de cálculos de manera controlada.

3.2.3 Filtros digitales: FIR e IIR

De acuerdo a Roads [9], un comité de ingenieros especializados en el procesamiento digital de señales dio esta definición de un filtro digital:

"Un filtro digital es un proceso computacional o algoritmo mediante el cual una señal digital o una secuencia de números (actuando como señal de entrada) se transforma en una segunda secuencia de números denominada, la señal digital de salida."

Según esta definición, todo sistema de tiempo discreto, como en la figura 3.1, es un filtro; sin importar la operación que el filtro realiza sobre la señal de entrada. El término filtro digital se utiliza en esta tesis en un sentido más específico: para detectar la envolvente de una señal de audio y tratar dinámicamente la función propia a cada procesamiento.

Para desarrollar tal filtro se requiere la funcionalidad de tres bloques básicos de construcción:

- El retraso del valor de muestreo por uno o varios periodos de muestreo.
- El escalamiento del valor de muestreo por un factor de ganancia.
- La mezcla (suma) de dos o más valores de muestreo.

La figura 3.2 muestra un filtro que retrasa una copia de la muestra actual de la señal de entrada, escala el nivel de la muestra retrasada por un factor de ganancia g y mezcla las señales directa y la retrasada. Es un filtro feedforward.

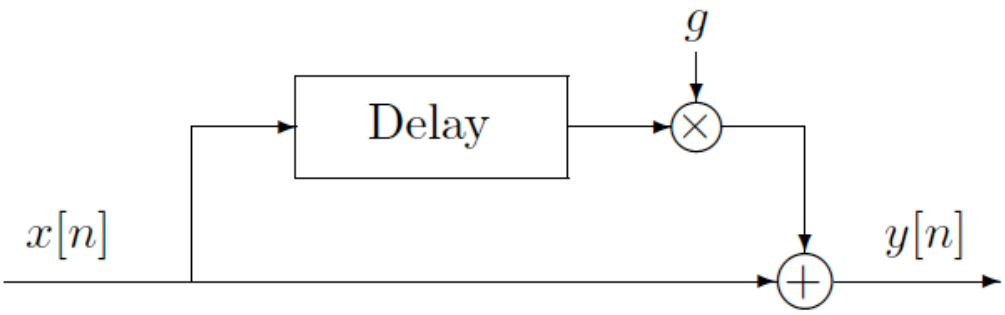


Figura 3.2. Entrada de entrada retrasada y adicionada a la salida (feedforward) [11]

Una fórmula en el dominio del tiempo (o en el dominio *n*) para calcular *y[n]* basada en la muestra de entrada presente y muestras pasadas de entrada o salida, o ambas, tiene el nombre de ecuación a diferencias. Dado un retraso de un período de muestreo, el filtro de la figura 3.2 puede ser descrito por la ecuación a diferencias

$$y[n] = x[n] + g \cdot x[n-1] \qquad \text{(Ec.3.4)}$$

en donde *x[n]* es la muestra actual de la señal de entrada y *x[n-1]* es la muestra previa (retrasada) de la señal de entrada. Esto se denomina un filtro de primer orden, ya que el retraso máximo utilizado en la ecuación 3.4 es un periodo de muestreo. Un filtro de segundo orden tiene un retraso máximo de dos periodos de muestreo, y así sucesivamente.

Si la señal de entrada para el filtro descrito en la ecuación 3.4 es un impulso

$$\delta[n] = \begin{cases} 1, n = 0 \\ 0, n \neq 0 \end{cases} \qquad \text{(Ec.3.5)}$$

la señal de salida, o respuesta impulsiva, tendrá un número finito de valores de muestreo diferentes de 0 – el impulso original se encontrará en *y[0]* seguido por el impulso retrasado y escalado en *y[1]*. Este filtro tipo feedforward es conocido como Filtro de Respuesta Finita (FIR). La ecuación 3.4 describe entonces un filtro FIR de primer orden.

Un filtro que retrasa una copia de la muestra actual de la señal de salida, escala el nivel de la muestra retrasada por un factor de ganancia g y mezcla las señales directas y retrasadas se muestra en la figura 3.3.

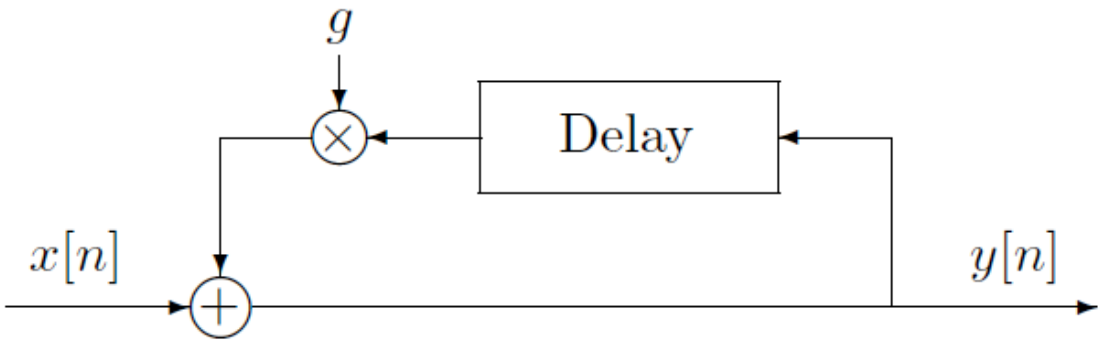


Figura 3.3. Señal de salida retrasada y adicionada a la entrada (feedback) [11]

El filtro tiene un camino de retroalimentación y se describe por la ecuación a diferencias

$$y[n] = x[n] + g \cdot y[n-1] \qquad \text{(Ec.3.6)}$$

siempre y cuando el retraso sea de un período de muestreo. La respuesta impulsiva de este filtro tendrá en teoría un número infinito de valores de muestreo no iguales a cero debido al camino de retroalimentación, el cual siempre alimenta la entrada con parte de la señal de salida. Este tipo de filtro de retroalimentación se conoce como un filtro de Respuesta Infinita de Impulso (IIR), y la ecuación 3.6 describe un filtro IIR de primer orden [11].

3.3 Procesamiento no lineal

3.3.1 Introducción

Dentro de la categoría de procesamiento no lineal se encuentran los algoritmos de efectos de audio utilizados para procesamiento de la dinámica de la señal, simulación de transistores de tubos, saturación y distorsión para guitarra y aplicaciones de grabación

de estudio y mejoradores sicoacústicos. Estos crean frecuencias armónicas o inarmónicas de manera intencional o no intencional que no están presentes en la señal de entrada. La distorsión armónica es causada por las no linealidades dentro del procesador de efectos. La mayoría de estos procesadores están controlados por una variedad de parámetros y, por medio de un medidor de nivel, escucha y monitorea simultáneamente la señal de salida. Se necesita mucha experiencia en grabación para obtener los resultados sonoros preferidos por la mayoría de los escuchas. La aplicación de estos aparatos de procesamiento de señal es un arte por sí mismo y, por supuesto, una de las principales herramientas para ingenieros de sonido y músicos [1].

Procesamiento no lineal o procesadores no lineales es el término usado para algoritmos de procesamiento de señal o aparatos para procesamiento de señal que entregan una señal de salida como la suma de señales senoidales, en los dominios análogos y digitales

$$y(n) = A_0 + A_1 \sin(2\pi f_1 Tn) + A_2 \sin(2 \cdot 2\pi f_1 Tn) + \ldots + A_N \sin(N \cdot 2\pi f_1 Tn),$$

(Ec.3.7)

si la señal de entrada es un sinusoide de amplitud conocida y frecuencia correspondiente a

$$x(n) = A_1 \sin(2\pi f_1 Tn). \quad \text{(Ec.3.8)}$$

Un sistema lineal entregara una señal de salida

$$y(n) = A_{out} \sin(2\pi f_1 Tn + \varphi_{out}) \quad \text{(Ec.3.9)}$$

la cual nuevamente es un sinusoide cuya amplitud es modificada por la respuesta de magnitud $|H(f_1)|$ de la función de transferencia de acuerdo con $A_{out} = |H(f_1)|$; y la fase de respuesta $\varphi_{out} = \varphi_{in} + H(f_1)$ es modificada por la fase $H(f_1)$ de la función de transferencia. Los diagramas de la figura 3.4 muestran señales de entrada y salida de un sistema lineal y no lineal para ilustrar la diferencia entre ambos sistemas. Una medida

de la distorsión armónica total da una indicación de la no linealidad del sistema. La distorsión armónica total se define como

$$THD = \sqrt{\frac{A_2^2+A_3^2+\cdots+A_N^2}{A_1^2+A_2^2+\ldots+A_N^2}} \qquad \text{(Ec.3.9)}$$

(donde A_1 es la fundamental y A_N las armónicas) la raíz cuadrada de la relación de la suma de potencias de todas las frecuencia armónicas que estén sobre la frecuencia fundamental dividido para la suma de potencias todas las frecuencias armónicas incluida la frecuencia fundamental [1].

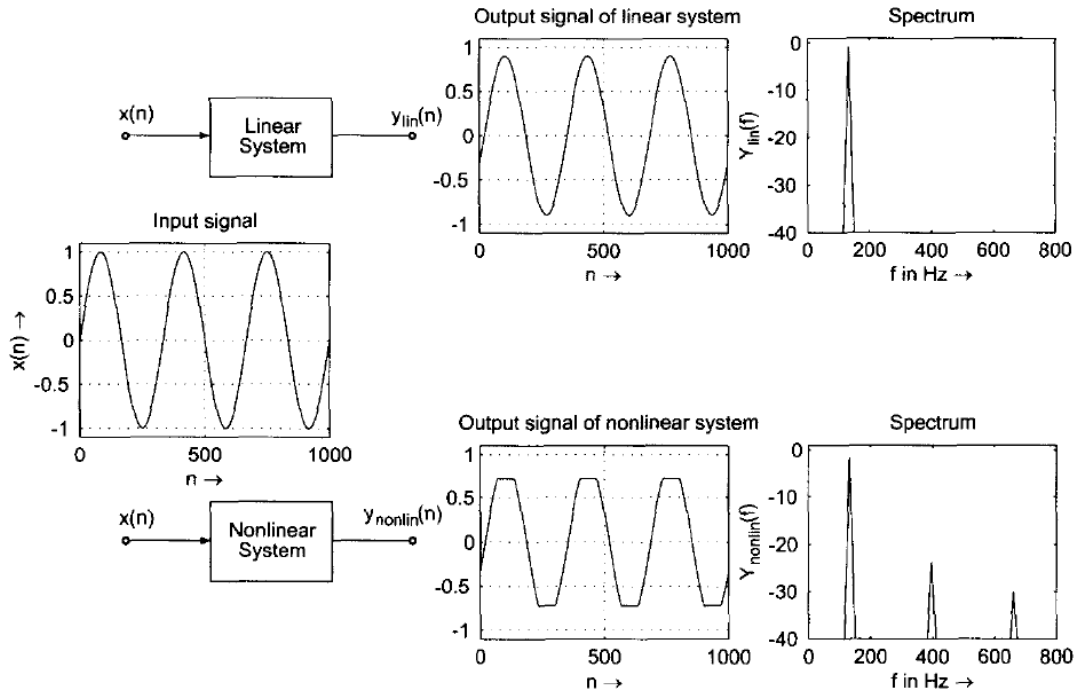


Figura 3.4. Señales de entrada y salida de un sistema lineal y no lineal.

Discutiremos el uso de procesamiento no lineal en una de las principales categorías de efectos musicales. Esta categoría consiste en los controladores de rango dinámico, donde el principal propósito es el control de la envolvente de la señal de acuerdo a ciertos parámetros de control. La cantidad de distorsión armónica creada por

estos algoritmos de control, se debe mantener tan baja como sea posible. Los algoritmos de procesamiento dinámico serán presentados en el capítulo 4.

3.3.2 Procesamiento dinámico de señales de audio

El procesamiento dinámico de señales de audio está basado en un sistema de detección de amplitud/nivel a veces llamado seguidor de envolvente, un algoritmo para obtener un factor de ganancia del resultado del seguidor de envolvente y un multiplicador del peso de la señal de entrada.

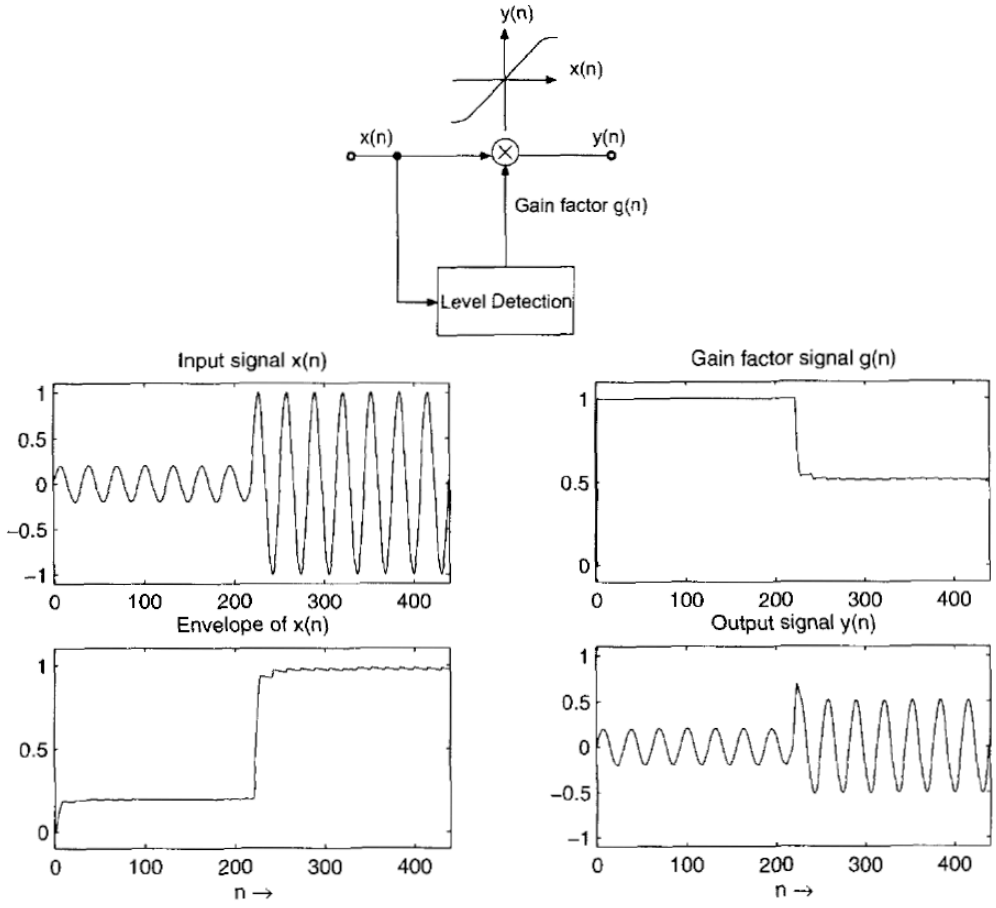


Figura 3.5. Diagrama de bloques de un procesador no lineal con detector de envolvente.

Los gráficos inferiores muestran la señal de entrada *x[n]*, la envolvente de *x[n]* el factor de ganancia que se deriva *g[n]* y la señal de salida *y[n]*.

El seguidor de envolvente calcula la media de valores absolutos *y[n] = f(x[n])* a lo largo de un intervalo de tiempo predefinido. La relación entre salida y entrada es normalmente descrita por la curva característica *y[n] = f(x[n])* como muestra la figura 3.5. Para el ejemplo dado la señal de salida está limitada a $y(n) = \mp 0.5$ para $|x(n)| > 0.5$ y a *y[n]=x[n]* para $|x(n)| \leq 0.5$. El camino inferior que consiste en un detector de envolvente y el procesamiento subsecuente para llegar al factor de ganancia *g[n]*es usualmente llamado "Side Chain"(ruta paralela en la cadena de procesamiento). Normalmente el factor de ganancia se deriva de la señal de entrada pero el camino paralelo también puede ser conectado a otra señal para controlar el factor de ganancia de la señal de entrada.

Una descripción detallada de un controlador de rango dinámico se muestra en la figura 3.6. Éste consiste en un camino directo para retrasar la señal de entrada y una ruta paralela (side chain path). La ruta paralela realiza una medición de nivel y calcula el factor de ganancia, el cual es luego utilizado como factor de ganancia para la señal de entrada retrasada. La medición de nivel es seguida por una función estática y el ajuste de tiempo para ataque y liberación. Aparte de las señales de tiempos, *x[n], f[n], g[n]* y *y[n]* se distinguen sus correspondientes niveles de señales *X, G* y *Y*. Estos valores de nivel corresponden al logaritmo del valor eficaz (valor RMS) o valor pico de las señales de tiempo de acuerdo a *X=20log(x)*. La multiplicación

$$y[n] = g[n] \cdot x[n - D] \qquad \text{(Ec.3.10)}$$

en la salida del controlador de rango dinámico puede ser considerada como una suma en el dominio logarítmico. Esto significa *Y = X + G* en dB. El cálculo del factor de ganancia de tiempo *g[n]* normalmente se realiza con una representación del nivel logarítmico ya que la respuesta del oído humano es logarítmica. El retraso de *D* muestras en el camino directo permite el retraso de tiempo del procesamiento paralelo el

cual consiste principalmente de la medición del nivel y los ajustes de tiempo de ataque y liberación.

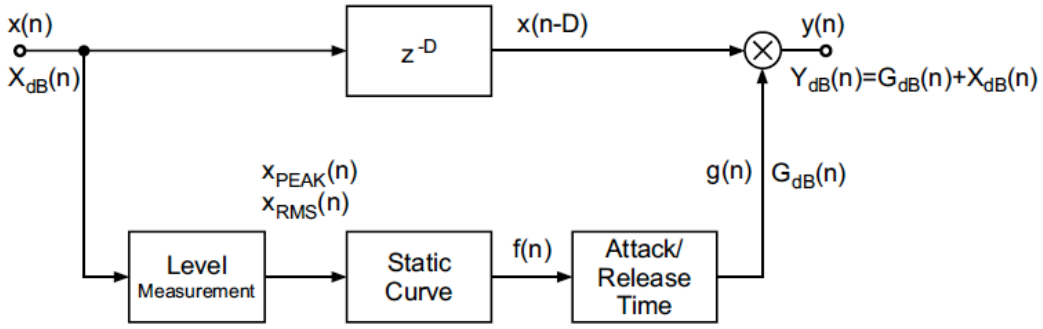


Figura 3.6. Representación de un controlador de rango dinámico [2]

La relación entre nivel de entrada y nivel de ponderación se define por una curva de nivel estático. Un ejemplo de esta curva estática se da en la figura 3.7. Aquí el nivel de salida y el nivel de ponderación se dan como funciones del nivel de entrada. En esta representación los puntos de inflexión representan los umbrales (threshold) para limitación (umbral de limitador), compresión (CT), expansión (ET) y compuerta de ruido (NT). Con la ayuda de un limitador, el nivel de salida se limita cuando el nivel de entrada excede nivel del umbral limitador (LT). Todos los niveles de entrada sobre este umbral llevan resultan en un nivel de salida constante. El compresor asigna un cambio en el nivel de entrada a un determinado cambio inferior de la señal de salida. A diferencia de un limitador, un compresor aumenta el volumen de la señal de audio. El expansor incrementa los cambios del nivel de entrada para expandir la dinámica del nivel de salida. La compuerta de ruido se utiliza para suprimir las señales de nivel bajo, para reducción de ruido y también para efectos de sonido como la eliminación de la reverberación de una habitación. Cada umbral utilizado en una parte específica de la curva estática se define como el límite inferior para limitador y compresor, y límite superior para expansor y compuertas de ruido.

Para la descripción de la función estática, se utilizan dos nuevos parámetros, llamados el factor de pendiente (Slope factor) *S* y el factor de compresión, *R.* Su relación se muestra a continuación en la ecuación 3.11.

$$R = \frac{1}{S} \qquad \text{(Ec.3.11)}$$

El factor de compresión *R* (ratio) representa la relación de cambio entre el nivel de entrada y el nivel de salida. Con la ayuda de la figura 3.7 se pueden derivar las ecuaciones de las funciones estáticas de cada procesamiento. Esto lo veremos más adelante en este capítulo cuando se detalle el funcionamiento y características de cada procesamiento [1].

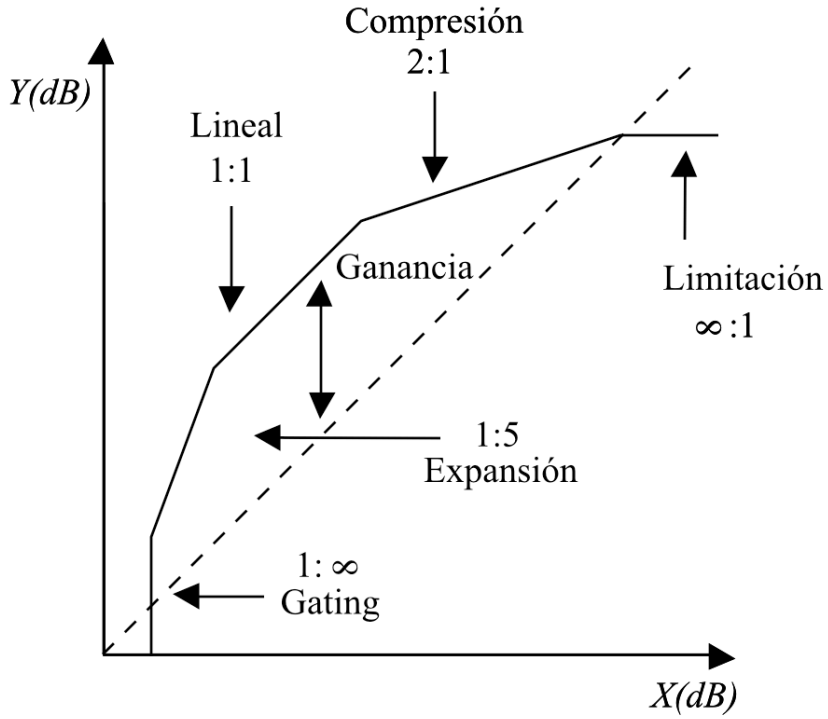


Figura 3.7. Gráfica que muestra las distintas zonas de operación de los diferentes tipos de procesamientos dinámicos

Valores típicos de para factor de compresión para las cuatro regiones de operación son:

Limitador	$R = \infty$
Compresor	$1 < R < \infty$
Región lineal	$R = 1$
Expansor	$0 < R < 1$
Compuerta de ruido	$R = 0$

Tabla 1. Valores típicos para el factor de compresión R para las cuatro regiones de operación [1]

3.3.3 Comportamiento dinámico

El comportamiento dinámico de un controlador de rango dinámico se ve determinado por el método de medición de nivel: tiempo de ataque y tiempo de liberación para medición de valor pico y tiempo medio para la medida de valor eficaz o valor RMS – *TAV* (Time Averager); y ajustado posteriormente con tiempos de ataque y liberación específicos que se pueden alcanzar mediante la utilización de determinados sistemas.

Además de la curva estática de control de rango dinámico el comportamiento dinámico, en términos de tiempo de tiempos los de ataque y liberación, juegan un rol importante en la calidad del sonido. La rapidez del control de rango dinámico también depende de las medidas de valor eficaz y valor pico.

3.3.4 Medición de nivel

La medición de nivel se puede realizar mediante los sistemas mostrados en las figuras 3.8 y 3.9. Para la medición de valor pico se comparan el valor absoluto de

entrada con el valor de pico $x_{PEAK}[n]$. Si el valor absoluto es mayor que el valor de pico, la diferencia se pondera con el coeficiente de ataque de tiempo *AT* y sumada a $(1 - AT) \cdot x_{PEAK}[n-1]$. Para este caso, $|x[n]| > x_{PEAK}[n-1]$ obtenemos la siguiente ecuación a diferencias (ver figura 3.8)

$$x_{PEAK}[n] = (1 - AT)x_{PEAK}[n-1] + AT \cdot |x[n]| \qquad \text{(Ec.3.12)}$$

con la función de transferencia

$$H(z) = \frac{AT}{1-(1-AT)z^{-1}}. \qquad \text{(Ec.3.13)}$$

Si el valor absoluto de la entrada es menor al valor de pico $|x[n]| \leq x_{PEAK}[n-1]$ el nuevo valor de pico se está dado por la ecuación 3.14

$$x_{PEAK}[n] = (1 - RT) \cdot x_{PEAK}[n-1] \qquad \text{(Ec.3.14)}$$

con el coeficiente de tiempo de liberación *RT*. Para este caso, la función de transferencia definida por la ecuación 3.15 es válida

$$H(z) = \frac{1}{1-(1-RT)z^{-1}} \qquad \text{(Ec.3.15)}$$

es válida. Para el caso de ataque utilizamos la función de transferencia (Ec.3.13) con el coeficiente *AT* y para el caso de liberación la función de transferencia (Ec.3.15) con el coeficiente *RT*. Los coeficientes se dan respectivamente por las ecuaciones 3.16 y 3.17 (revisar sección 3.3.6).

$$AT = 1 - exp\left(\frac{-2.2T_s}{t_a/1000}\right) \qquad \text{(Ec.3.16)}$$

$$RT = 1 - exp\left(\frac{-2.2T_s}{t_r/1000}\right) \qquad \text{(Ec.3.17)}$$

En donde el tiempo de ataque t_a y el tiempo de liberación t_r se dan en mili segundos (el intervalo de muestreo T_s). Alternando entre tipos de filtros se pueden alcanzar respuestas de ataque rápidas para señales de entrada en aumento, y respuestas lentas de decaimiento para amplitudes de señal de entrada que decrecen en nivel [2].

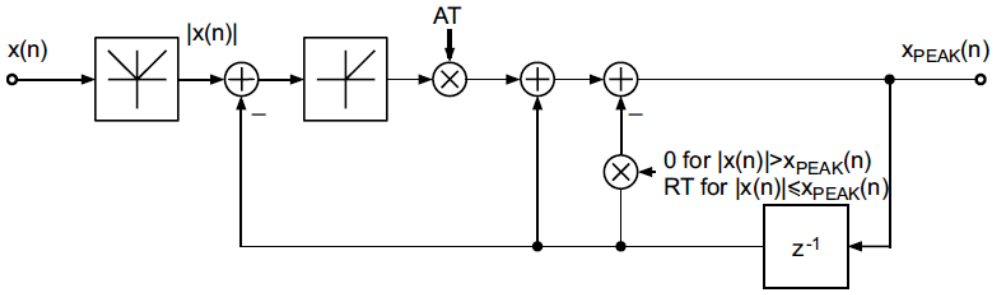


Figura 3.8. Medición de nivel pico [2]

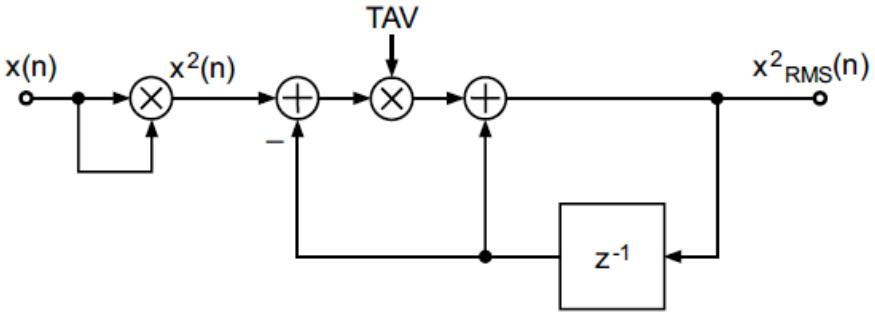


Figura 3.9. Medición de nivel RMS [2]

El cálculo del valor eficaz, *Xrms*, que se indica en la ecuación 3.17,

$$x_{RMS} = \sqrt{\frac{1}{N}\sum_{i=0}^{N-1} x^2[n-i]} \qquad \text{(Ec.3.17)}$$

de N muestras de entrada puede ser determinado por una formulación recursiva. La medición de valor eficaz mostrada en la figura 3.9 utiliza la raíz de la entrada y realiza filtraje con un filtro pasabajos de primer orden. El coeficiente del filtro *TAV*, que corresponde a la ecuación 3.18,

$$TAV = 1 - exp\left(\frac{-2.2T_s}{t_M/1000}\right) \qquad \text{(Ec.3.18)}$$

se determina de acuerdo al cálculo de la constante de tiempo que veremos en la sección 3.3.6; en donde t_M es el tiempo en milisegundos. La ecuación de diferencias de este sistema se muestra en la ecuación 3.19

$$x_{RMS}[n] = (1 - TAV)x_{RMS}[n-1] + TAV \cdot x[n] \qquad \text{(Ec.3.19)}$$

con su respectiva función de transferencia representada por la ecuación 3.20

$$H(z) = \frac{TAV}{1-(1-TAV)z^{-1}}. \qquad \text{(Ec.3.20)}$$

3.3.5 Alisaje del factor de ganancia

Tiempos de ataque y liberación pueden ser implementados por el sistema mostrado en la figura 3.10. El coeficiente de ataque *AT* o el coeficiente de liberación *RT* se obtienen al comparar el factor de entrada actual con el anterior. Una pequeña curva de histéresis determina si el factor de control se encuentra en el estado de ataque o liberación y por lo tanto determina el coeficiente a ser usado. El sistema también sirve para alisar la señal de control. La ecuación diferencial se da por la ecuación 3.21

$$g[n] = (1 - k) \cdot g[n-1] + k \cdot f[n], \qquad \text{(Ec.3.21)}$$

con $k = AT$ o $k = RT$ y la correspondiente función de transferencia se deriva la ecuación

$$H(z) = \frac{kt}{1-(1-kT)z^{-1}}. \qquad \text{(Ec.3.22)}$$

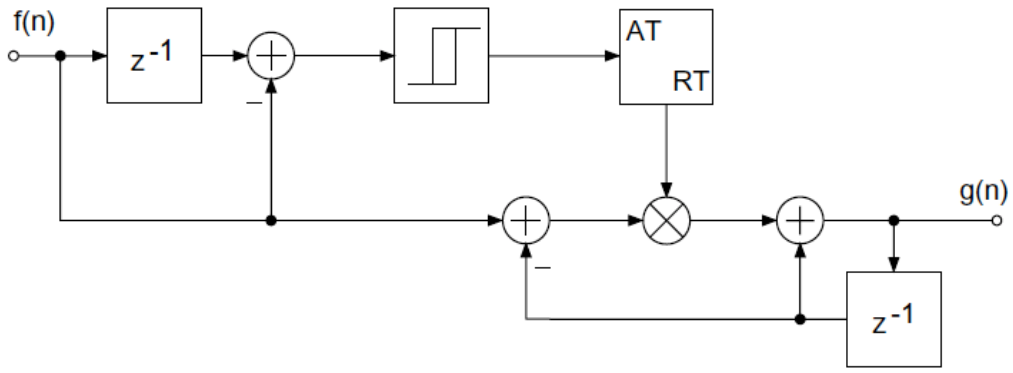


Figura 3.10. Implementación de tiempos de ataque y liberación para alisaje del factor de ganancia [2]

3.3.6 Constantes de tiempo

Si la respuesta escalón de un sistema de tiempo continuo es como el de la ecuación 3.23

$$g(t) = 1 - e^{-t/\tau}, \tau = constante\ de\ tiempo \quad \text{(Ec.3.23)}$$

entonces muestreando la respuesta escalón se obtiene la respuesta del escalón en tiempo discreto que vemos en la ecuación 3.24

$$g[nT_S] = \varepsilon[nT_S] - e^{-nT_S/\tau} = 1 - z_\infty^n,\ z_\infty^n = e^{-T_S/\tau}. \quad \text{(Ec.3.24)}$$

La transformada Z nos da

$$G(z) = \frac{z}{z-1} - \frac{1}{1-z_\infty z^{-1}} = \frac{1-z_\infty}{(z-1)(1-z_\infty z^{-1})}. \quad \text{(Ec.3.25)}$$

De la definición de tiempo de ataque $t_a = t_{90} - t_{10}$, derivamos

$$0.1 = 1 - e^{-t_{10}/\tau} \leftarrow t_{10} = 0.1\tau, \quad \text{(Ec.3.26)}$$

$$0.9 = 1 - e^{-t_{90}/\tau} \leftarrow t_{90} = 0.9\tau. \quad \text{(Ec.3.27)}$$

La relación entre tiempo de ataque t_a y la constante de tiempo τ de la respuesta de escalón se obtiene por la ecuación 3.28

$$\frac{0.9}{0.1} = e^{(t_{90}-t_{10})/\tau}$$

$$\ln(0.9/0.1) = (t_{90} - t_{10})/\tau \qquad \text{(Ec.3.28)}$$

$$t_a = t_{90} - t_{10} = 2.2\tau.$$

Entonces, el polo se calcula como la ecuación 3.29

$$z_\infty = e^{-2.2T_S/t_a}. \qquad \text{(Ec.3.29)}$$

Un sistema para implementar la respuesta de escalón dada se obtiene como resultado de la relación entre la transformada Z de la respuesta impulsiva y la transformada Z de la respuesta escalón, como lo vemos en la ecuación 3.30

$$H(Z) = \frac{z-1}{z} G(z). \qquad \text{(Ec.3.30)}$$

La función de transferencia ahora puede ser escrita como:

$$H(z) = \frac{(1-z_\infty)z^{-1}}{1-z_\infty z^{-1}} \qquad \text{(Ec.3.31)}$$

con el polo $z_\infty = e^{-2.2T_S/t_a}$ ajustando el tiempo de ataque, liberación y filtraje. Para los coeficientes de los filtros de tiempo constante correspondientes, el caso de ataque se obtiene por (Ec.3.16), el caso de liberación por la ecuación (Ec.3.17) y el caso de filtraje por (Ec.3.18). La figura 3.11 muestra un ejemplo en el cual la línea punteada representa el tiempo t_{10} y t_{90} [2].

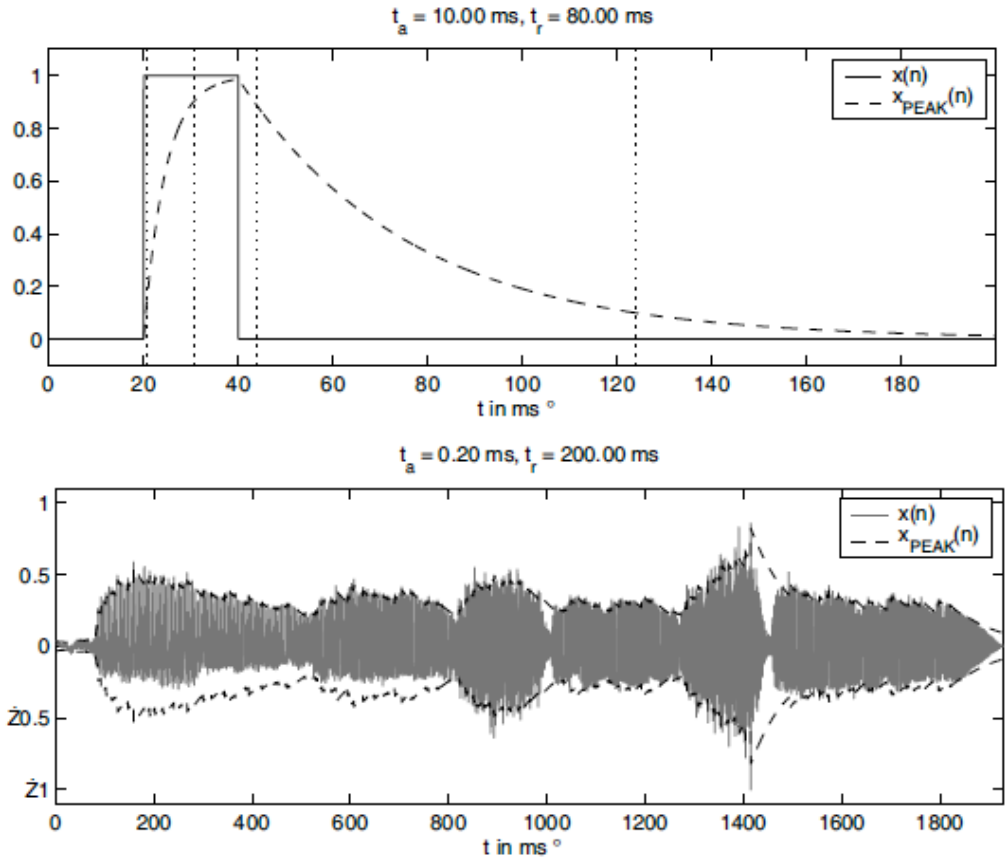


Figura 3.11. Comportamiento de ataque y liberación para filtros de tiempo constantes [2]

3.3.7 Limitador

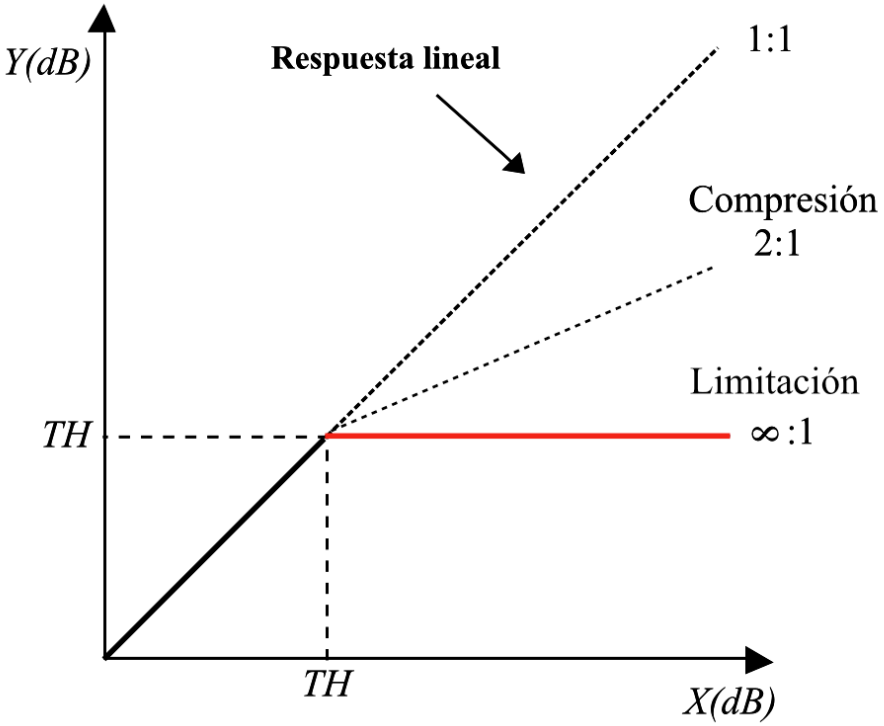


Figura 3.12. Gráfica que muestra la función de transferencia de un compresor

La figura 3.12 representa la función de transferencia de un limitador. El limitador es probablemente el más fácil y más común procesamiento dinámico. Prácticamente cualquier señal de audio que sea escuchada fuera de un estudio de grabación ha sido pasada por un limitador en algún punto de la cadena de audio – audio en TV, radio y especialmente en CDs. Esta persistencia tiene un peculiar resultado cultural que es que cada vez nuestros oídos se habitúan más a limitación y compresión excesiva y asistimos ahora a una competencia de niveles de escucha. Inicialmente este fenómeno se daba solo en las radios ya que unas estaciones querían tener más nivel que otras. Lamentablemente, el mismo fenómeno se está reproduciendo para los CDs. Las figuras 3.13 y 3.14 muestran como ha evolucionado este fenómeno y vemos claramente que las producciones contemporáneas son sometidas a cantidades brutales de limitación

y compresión. Esto es el resultado de la extraña tendencia del oído humando a creer que lo que suena más fuerte es de mejor calidad.

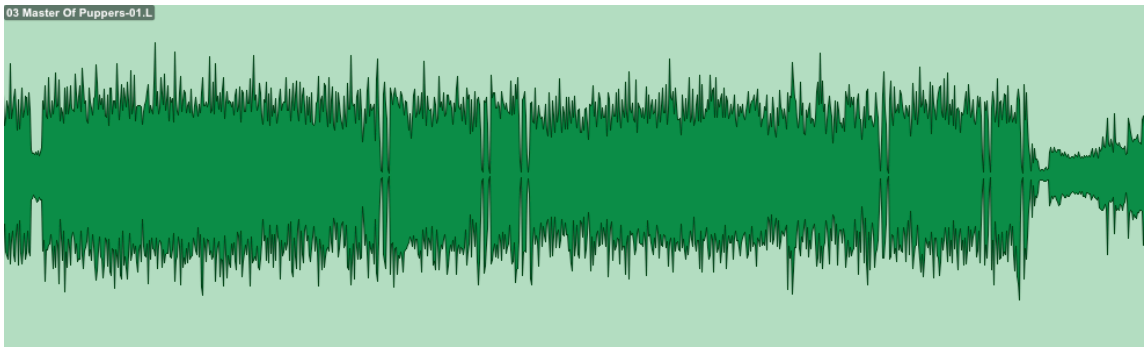


Figura 3.13 Nivel de *Master of Puppets* de Metallica en 1986

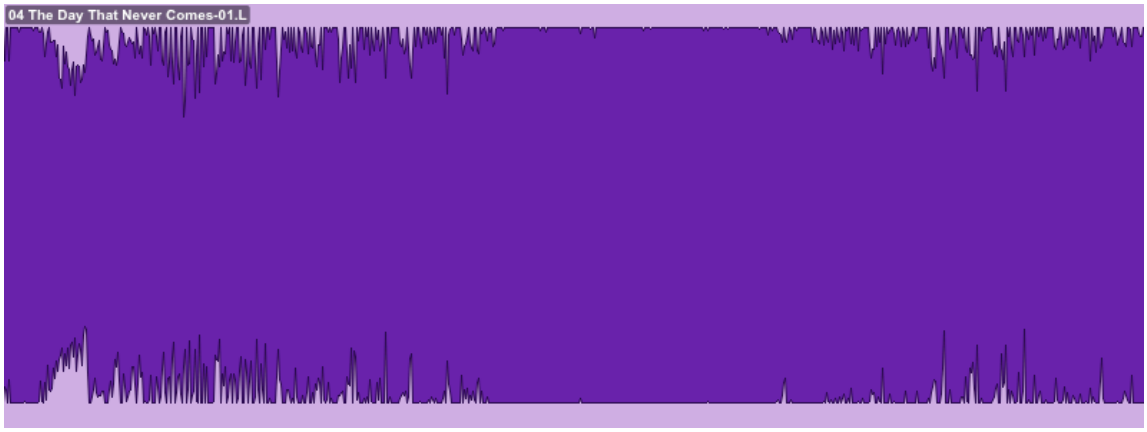


Figura 3.14 Nivel de *The Day That Never Comes* de Metallica en el 2008

Todo medio de grabación y transmisión tiene un límite dinámico dentro del cual la señal de audio puede oscilar libremente, es decir un límite sobre el cual la señal se distorsiona. Un limitador ayuda a que la señal de audio se mantenga dentro de esos límites.

Las unidades funcionales de un limitador se muestran en la figura 3.15. El propósito del limitador es proveer control sobre picos de la señal y cambiar la dinámica de la señal lo menos posible. El limitador hace uso de la medida del valor pico y deberá reaccionar muy rápidamente una vez que el valor pico sobrepasa el umbral del limitador.

Valores típicos para los parámetros de un limitador son $t_{AT} = 0.02ms$ y $t_{RT} = 130ms$ para la medida de valor pico y $t_{AT} = 10ms$ y $t_{RT} = 100ms$ para el ajuste de tiempo de ataque y liberación. El ataque y liberación rápidos de un limitador permiten la reducción del volumen tan pronto como la señal cruza el umbral. Al reducir los niveles picos se puede dar más nivel o volumen a la señal total. Aparte de limitar señales de instrumentos individuales, este procesamiento se usa también en la mezcla final de una aplicación multicanal.

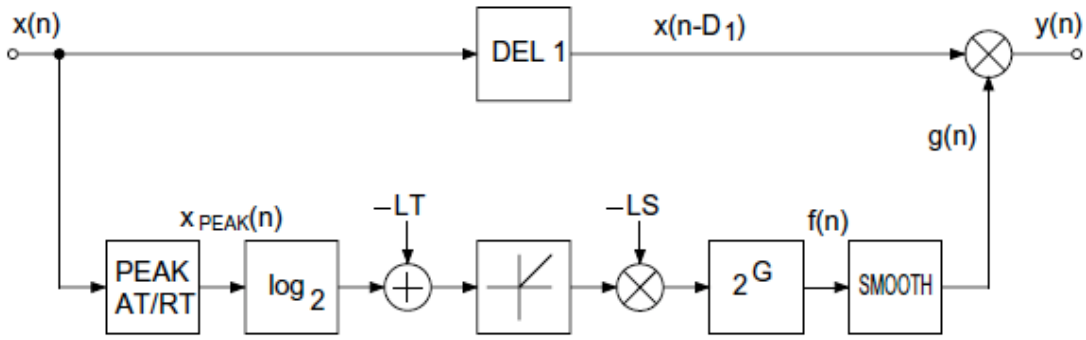


Figura 3.15. Limitador

La figura 3.16 muestra el comportamiento de la señal dentro de una configuración de limitador [2]. En esta figura es importante recalcar el bloque de medición de nivel. Para los limitadores se debe realizar una medición de nivel pico.

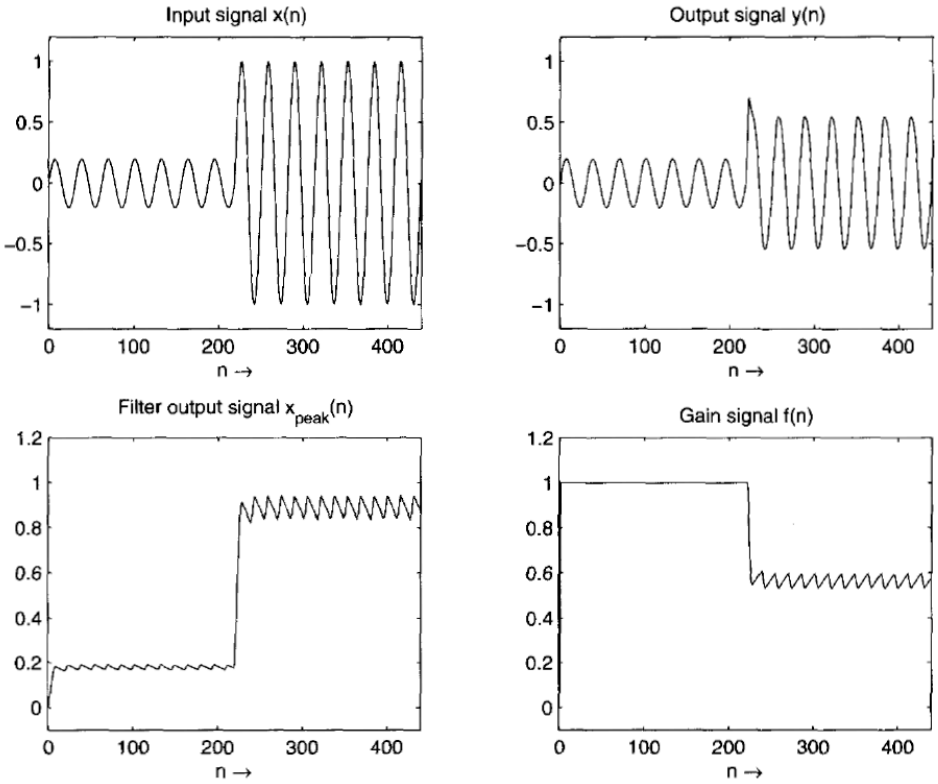


Figura 3.16. Formas de onda de un limitador [1]

3.3.8 Compresor

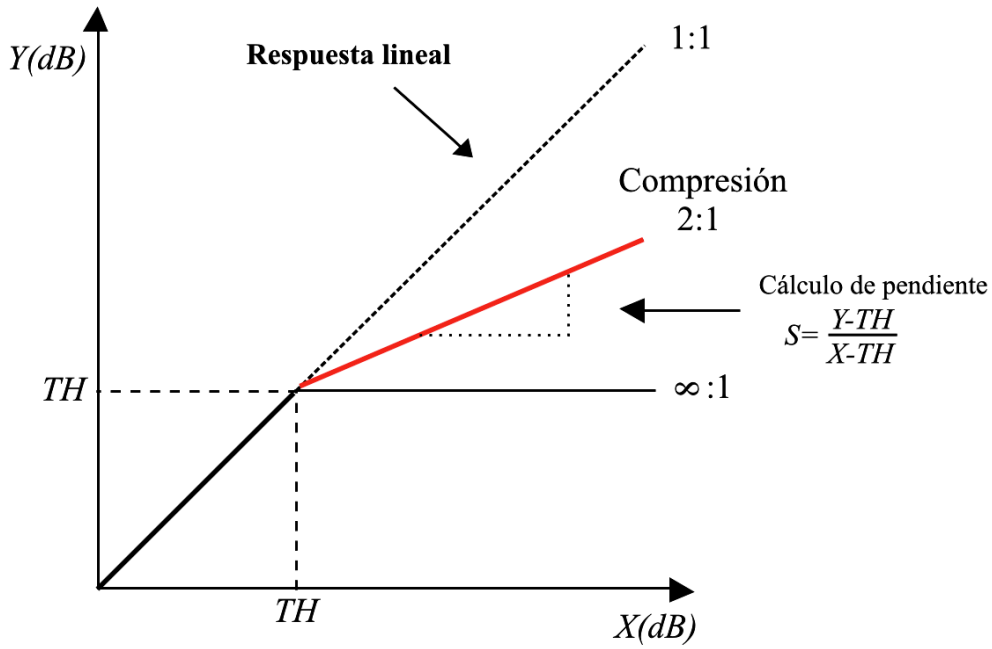


Figura 3.17. Gráfica que muestra la función de transferencia de un limitador

La figura 3.17 representa la función de transferencia de un compresor. Un controlador de rango dinámico para compresión se muestra en la figura 3.18. El cálculo del factor de ganancia se basa en una medida RMS y algunos cálculos en el dominio logarítmico. Los parámetros típicos para compresores son $t_{TAV} = 0.01ms$ para la medida de valor eficaz y $t_{AT} = 5ms$ y $t_{RT} = 130ms$ para los ajustes de tiempo de ataque y liberación. La programación es similar a la implementación del limitador de la sección anterior.

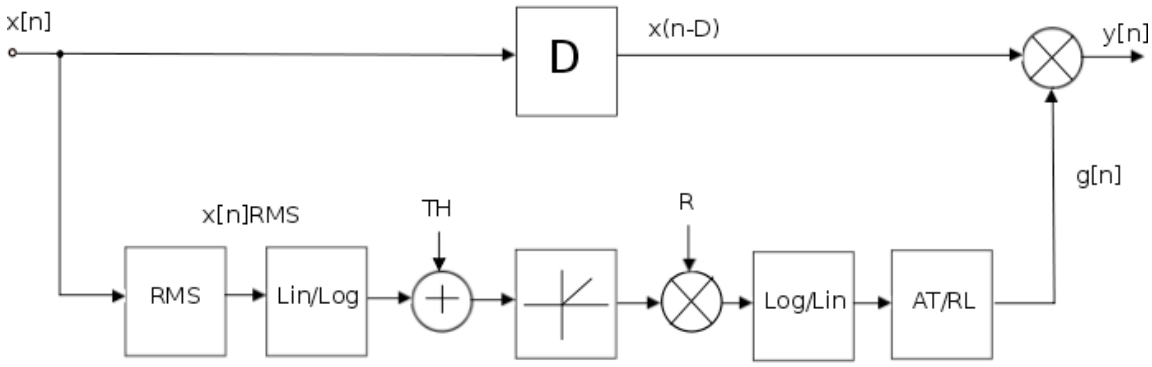


Figura 3.18. Compresor [1]

Los compresores son utilizados para reducir la dinámica de la señal de entrada. Los niveles bajos de la señal no son modificados y los niveles altos son reducidos de acuerdo a la curva estática la cual es determinada por el umbral del compresor y por la relación de compresión *R*. El resultado es que la diferencia entre los niveles bajos y altos de la señal es reducida y de esta manera se puede aumentar el nivel general de la señal, lo que hace que la intensidad de la señal sea mayor. En las figuras 3.19 y 3.20 podemos ver como la señal no es modificada mientras el nivel de la señal no sobrepase el umbral, una vez que la señal sobrepasa el umbral el compresor actúa haciendo un control de nivel rápido o lento de acuerdo a las constantes de liberación y ataque.

Una variación de configuración de un compresor es llamada “Voice over Compression” y se utiliza para comprimir señales que pasen por el compresor en función del nivel de otra señal que entra por un puerto adicional al sistema llamado “Side Chain”. El resultado es que la medición de nivel de una señal controla el rango dinámico de la otra señal. Esto se lo utiliza mucho, por ejemplo, en la radio, cuando los comentaristas hablan la música de fondo baja automáticamente de nivel. Esto es realizado precisamente por un compresor.

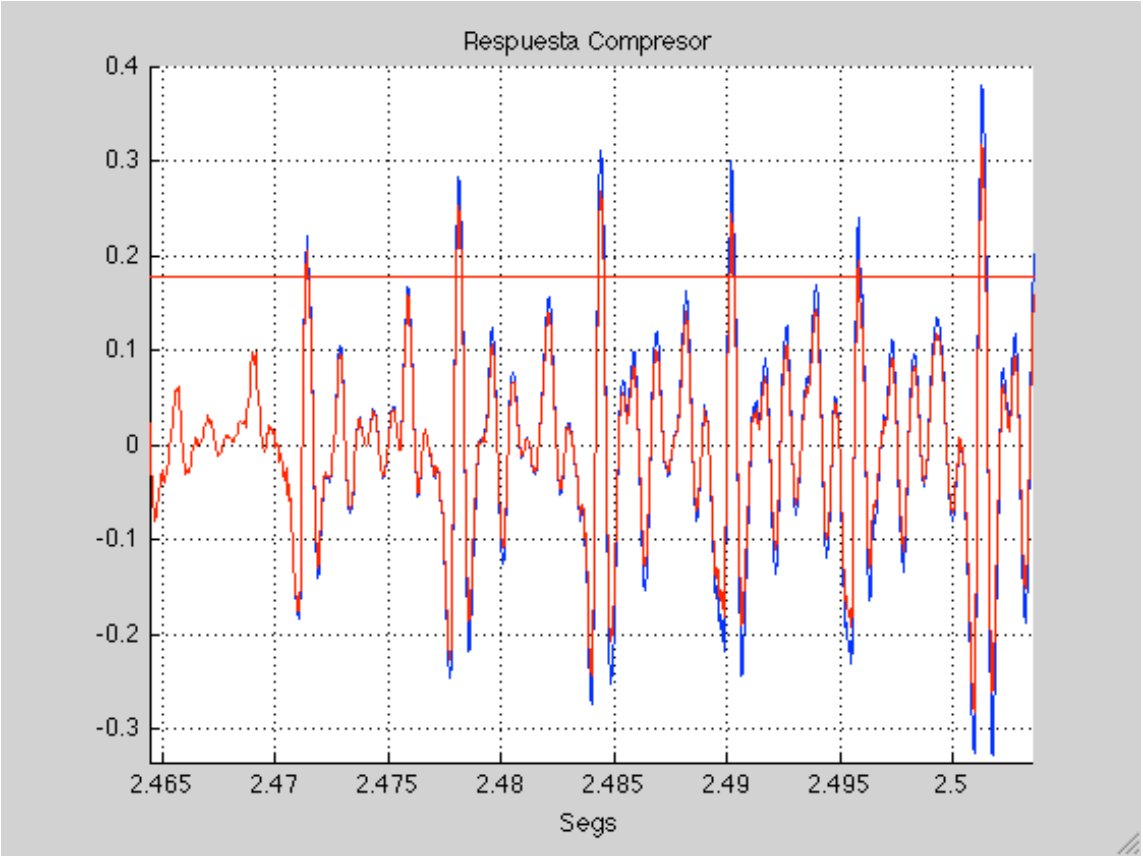


Figura 3.19. Respuesta del compresor en el momento que el nivel RMS de la señal cruza el umbral (Señal de entrada en azul y señal de salida en rojo).

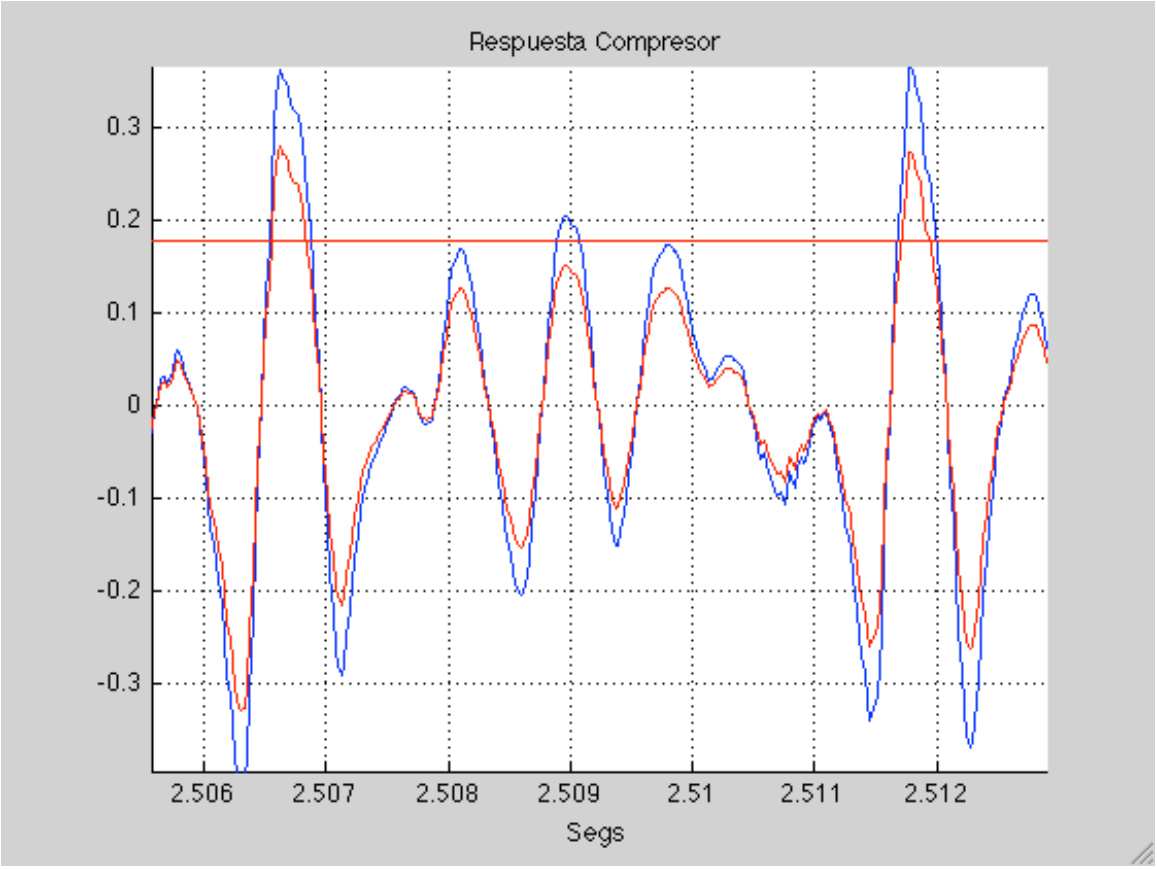


Figura 3.20. Zoom de la forma de la señal cuando el compresor está operando (Señal de entrada en azul y señal de salida en rojo).

3.3.9 Expansor

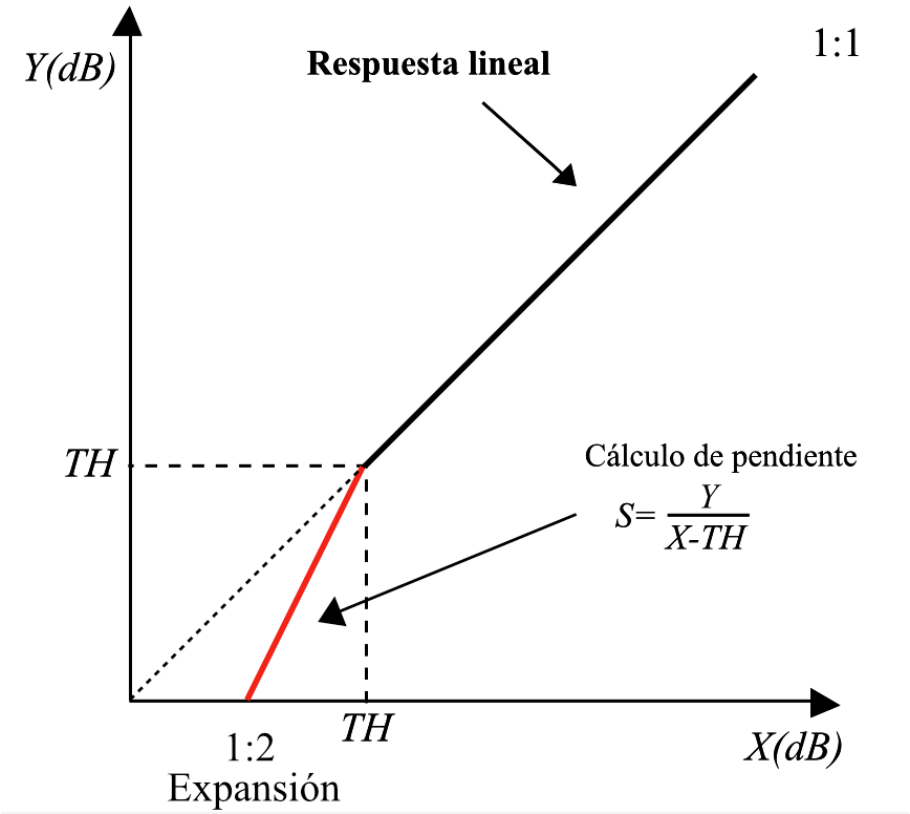


Figura 3.21. Gráfica que muestra la función de transferencia de un expansor

La figura 3.21 representa la función de transferencia de un expansor. Los expansores operan en señales de nivel bajo y aumentan la dinámica de la señal reduciendo estos niveles bajos. En la figura 3.21 podemos ver que el expansor se encarga de reducir el nivel de la señal de salida con respecto a la señal de entrada. Al disminuir los niveles de determinadas muestras (dependiendo del umbral del expansor) se consigue "expandir" la dinámica de la señal. Esto lo podemos ver claramente en la figura 3.22 donde constatamos que el nivel de la señal de salida es más bajo una vez que la señal de entrada está por debajo del umbral del expansor. Veremos más adelante que la compuerta de ruido no es más que una variación del expansor con un factor de expansión *R* (ratio) infinito.

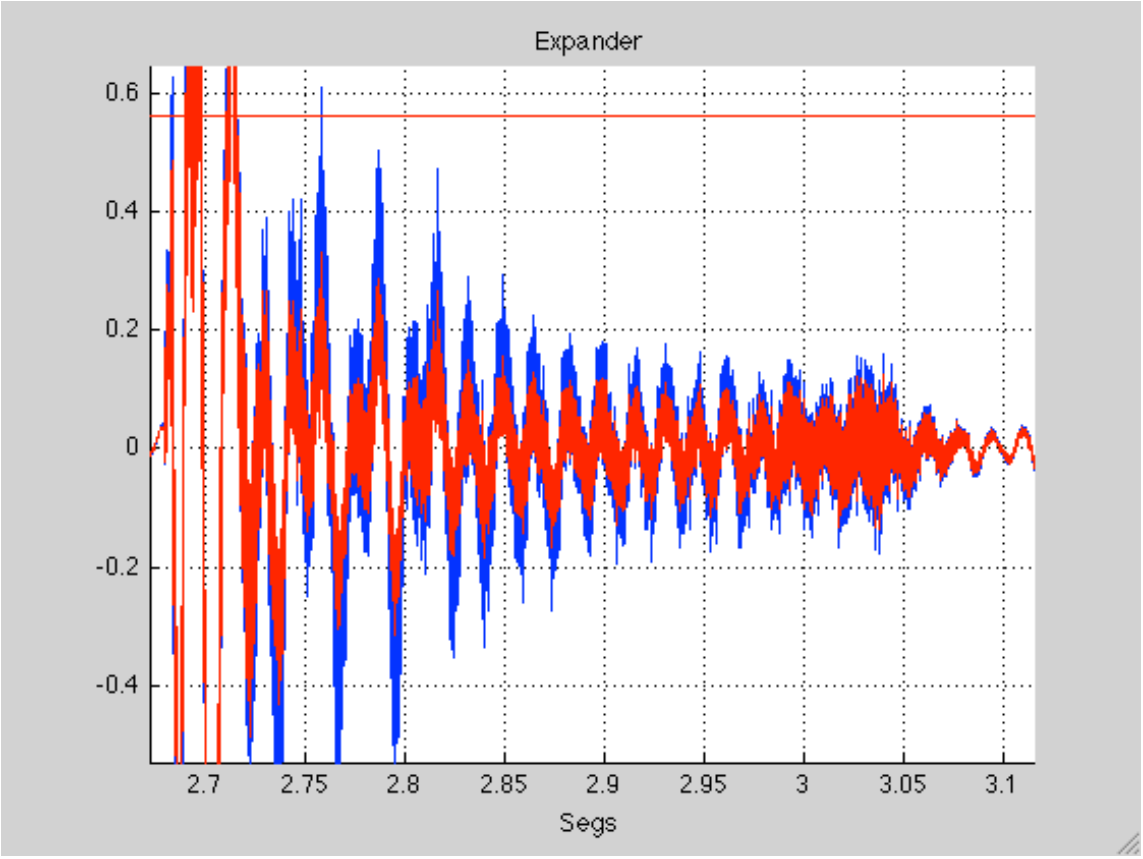


Figura 3.22. Respuesta del expansor en el momento que el nivel RMS de la señal pasa por debajo del umbral (Señal de entrada en azul y señal de salida en rojo).

3.3.10 Compuerta de ruido

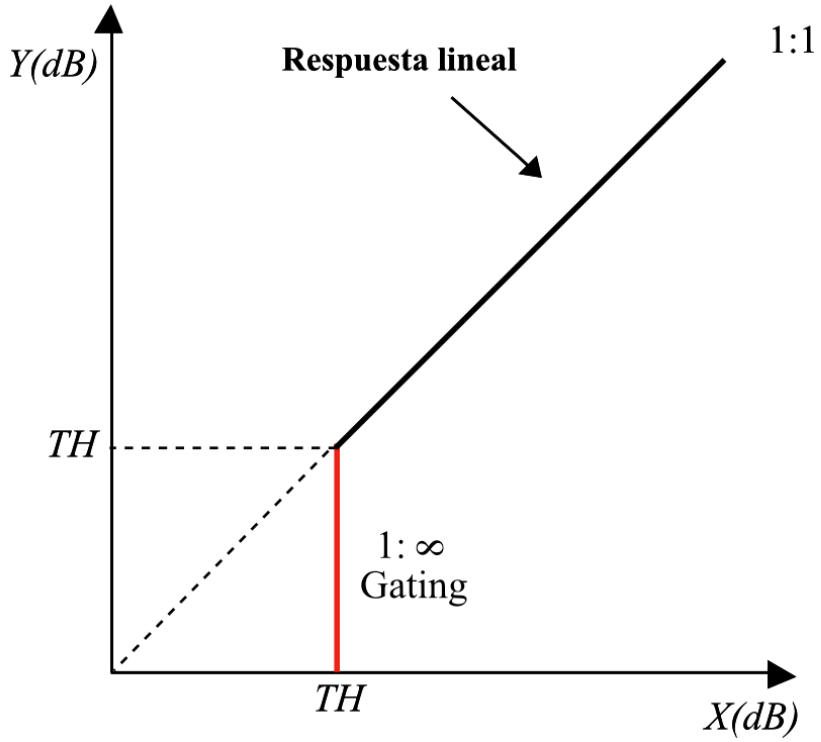


Figura 3.23. Gráfica que muestra la función de transferencia de una compuerta de ruido

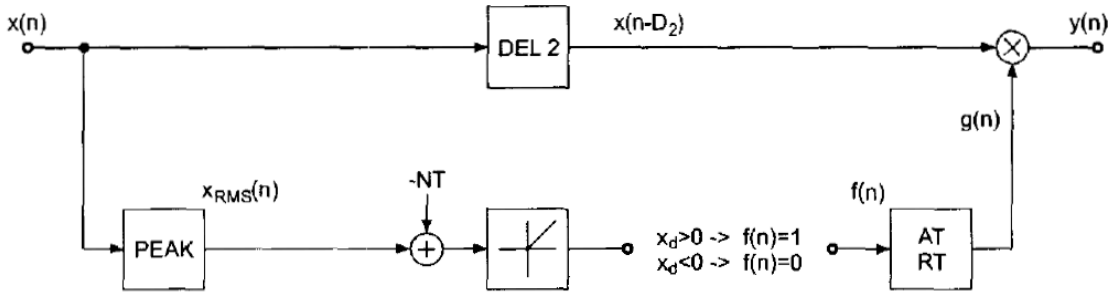


Figura 3.24. Compuerta de ruido [1]

La figura 3.23 representa la función de transferencia de una compuerta de ruido. Las compuerta de ruido realizan de cierta manera la operación opuesta a un limitador. Es la supresión de la señal a menos que está alcance cierto nivel; en otras palabras si la

señal de entrada sobrepasa el umbral de la compuerta se la permite pasar pero si este nivel cae por debajo del umbral entonces la señal es atenuada.

Su función es por lo general, reducir o suprimir el nivel de una señal que no contribuye a una mezcla, remover ruido entre partes y por lo general actuar como un mute automático. El manejo de sus constantes de tiempo hace que los resultados de una compuerta de ruido sean muy audibles o sutilmente audibles. Un ejemplo del empleo de compuerta de ruido es en la grabación de una batería. En la grabación de una batería se llega a tener hasta veinte micrófonos ya que existe muchas fuentes sonoras que hay que captar. Evidentemente al haber tanto ruido el micrófono del bombo va a captar el sonido el bombo pero también el de los platillos y la caja. Aunque presente, el nivel de estos otros elementos de la batería va a ser inferior que el nivel de bombo en ese micrófono, por lo tanto gracias a una compuerta de ruido bien configurada podemos eliminar ese sonido "parásito" en el micrófono deseado.

Las unidades funcionales de una compuerta de ruido se muestran en la figura 3.24. La decisión de activar la compuerta se basa en una medida de pico que conlleva a un fade in/hold/fade out del factor de ganancia con tiempos de ataque, hold y liberación apropiados. Si el nivel de entrada es menor que el umbral de activación de la compuerta de ruido entonces el factor de ganancia es de cero; y si el nivel de entrada es mayor al umbral entonces el factor de ganancia es de uno. La histéresis es un pequeño rango dentro del cual se decide el nivel del umbral al cual se abre la compuerta y el nivel del umbral al cual se cierra la compuerta. Es decir si tenemos una histéresis de 4dB esto quiere decir que si el umbral para activación de la compuerta es de -40dB la compuerta se cerrará cuando el nivel sea inferior a -44dB.

En la figura 3.25 vemos la forma de la curva de ganancia de una compuerta de ruido. El momento en que el nivel supera el umbral de activación (en negro), la

compuerta se abre lentamente de acuerdo al tiempo de ataque determinado por el usuario. Un vez que el nivel desciende por debajo del umbral determinado por la histéresis, la compuerta se queda abierta por el tiempo determinado por el tiempo de retención y luego se va cerrando en el tiempo de liberación.

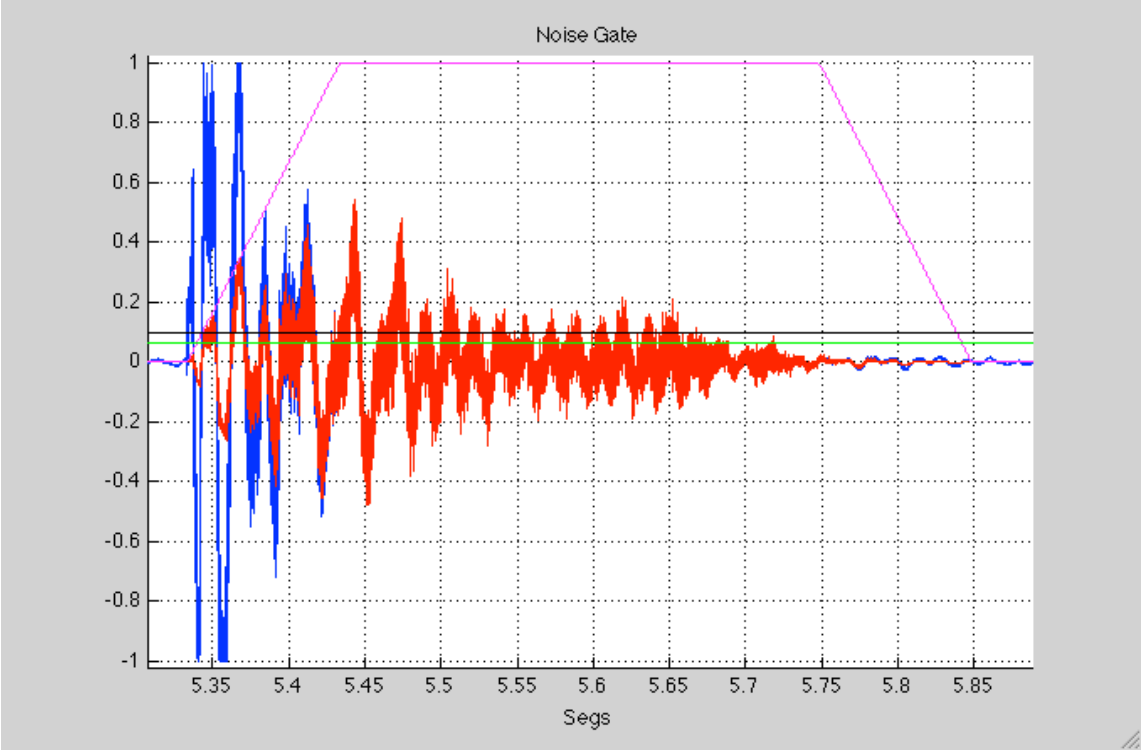


Figura 3.25. Respuesta de la compuerta de ruido (Señal de entrada en azul, señal de salida en rojo y ganancia de la compuerta en rosa).

3.3.11 Procesamiento estéreo

Para el procesamiento estéreo se necesita un factor de control común *g[n]*. Si se utilizan factores de control diferentes para ambos canales, el limitar o comprimir una de estas dos señales estéreo causa una alteración del balance del estéreo. La figura 3.26 muestra un sistema dinámico de estéreo en el cual la suma de las dos señales se utiliza para controlar el factor de control común. Los pasos subsiguientes para medir valores de pico y valores eficaces, tomar logaritmos, cálculo de curva estática, toma de tiempos de ataque y liberación, son los mismos que para el procesamiento en mono. El retraso en el camino directo de la señal debe ser el mismo para ambos canales.

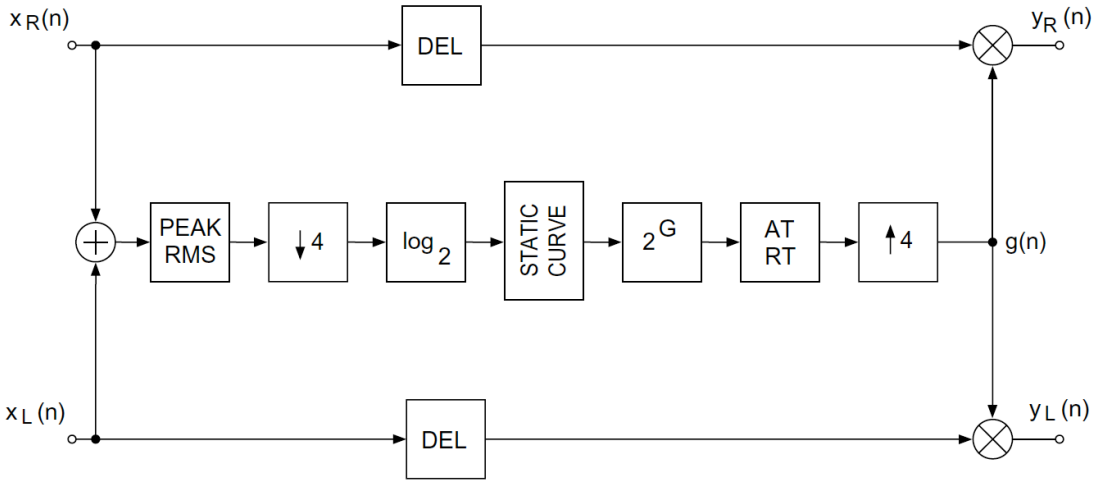


Figura 3.26. Sistema para tratamiento dinámico estéreo [2]

Capítulo 4 Prototipos en Matlab

Este capítulo pone en evidencia los resultados obtenidos en Matlab. Se realizaron cuatro prototipos correspondientes a cada tratamiento dinámico analizado en esta tesis. Se implementaron en los prototipo los algoritmos basados en los diagramas de Zölzer [2] para tratamientos digital de rango dinámico. El contenido de los archivos de Matlab (.m) se encuentra en los anexos 2.

Para las pruebas se utilizaron distintos tipos de señales de audio: grabación de voz, baterías y mezclas finales. Dependiendo del interés del tratamiento y de lo que se quería poner en evidencia, se variaron los parámetros para mostrar el buen desempeño de los algoritmos.

A continuación veremos uno por uno los procesadores y los resultados obtenidos al tratar la señales de audio con ellos. Para esto se mostrarán primeramente las señales de entrada y medición de nivel junto con el nivel del umbral (en rojo). Luego veremos la gráfica con las ganancias y la señal de salida una vez aplicada esta ganancia.

La función “wavread” de Matlab lee un archivo de audio tipo wave y lo codifica en valores entre -1.0 y 1.0. La representación gráfica de las señales de entrada y salida están por lo tanto representadas dentro de este rango de valores. Por otro lado la medición de nivel esta representada dentro de un rango de 0 a *log10(2)*. Esto se debe a que para evitar el *log10(0)* se debe sumar 1 al valor absoluto de la señal de entrada, el cual comprende valor entre 0.0 y 1.0.

4.1 Limitador

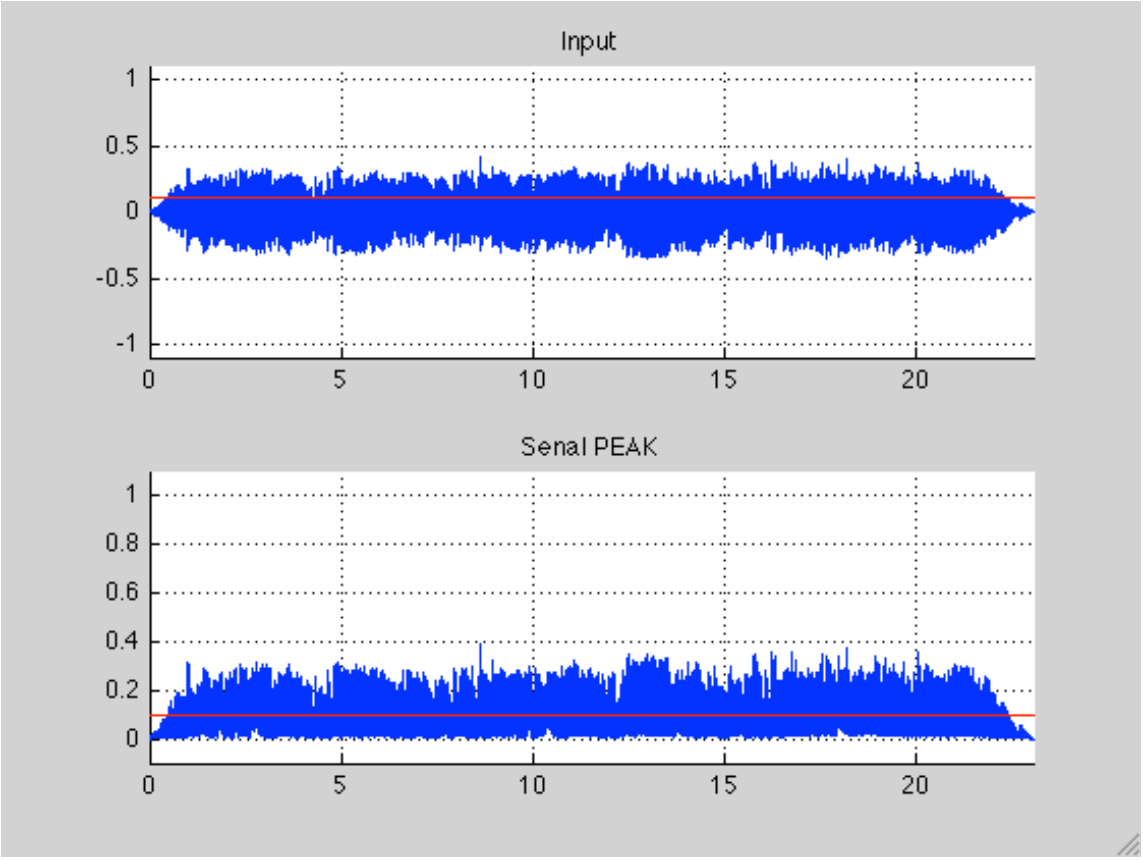


Figura 4.1. Señal de entrada y medición pico de nivel del limitador - Threshold=-20dB

En la figura 4.1 vemos la señal de una mezcla completa, es decir un segmento de una canción con diferentes instrumentos. Como vemos el nivel de la señal no es muy alto, por lo que debemos usar un nivel de threshold un poco bajo para que el limitador pueda actuar. Elegimos entonces un nivel de -20dB el cual está representado por la línea de color rojo. Esta representación está escalada para poder apreciar claramente el nivel del threshold. La segunda gráfica de la figura 4.2 representa la medición del nivel pico de la señal, como vemos su comportamiento es similar al comportamiento de la señal de entrada, lo que es normal.

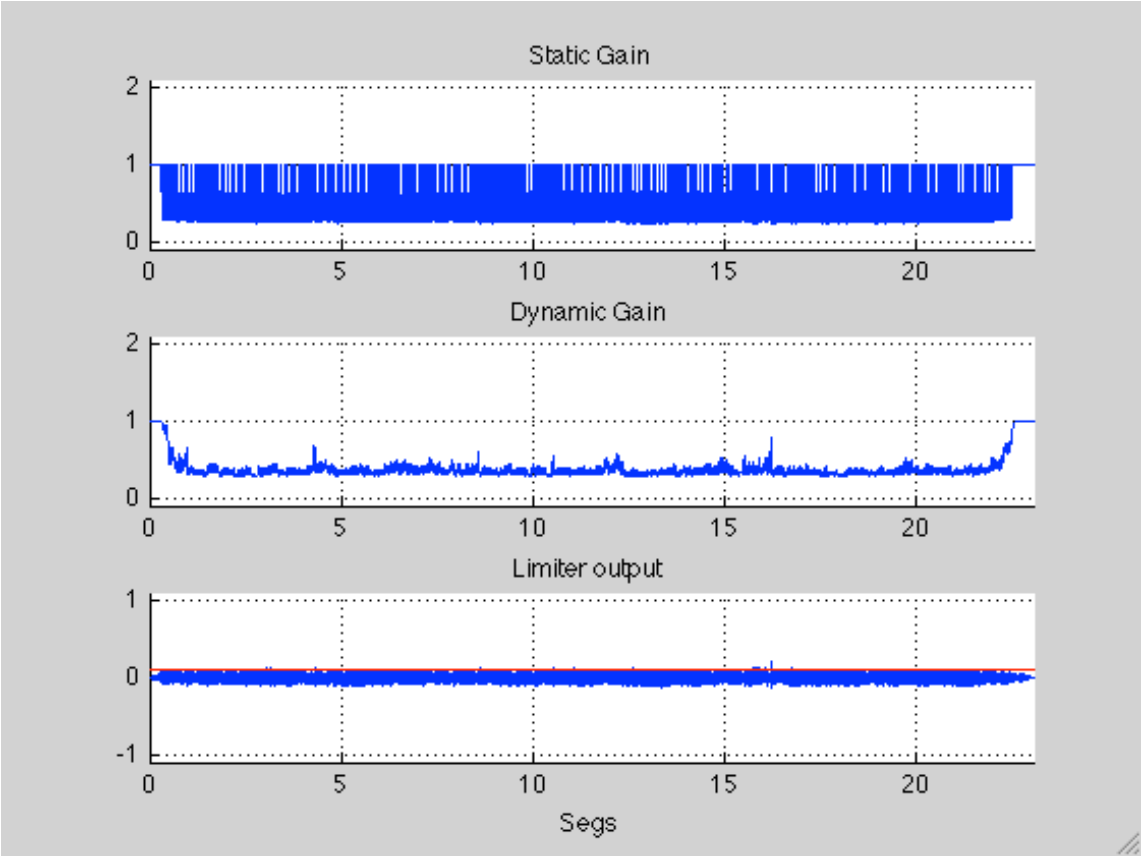


Figura 4.2. Limitador – Release=100ms/Attack=10ms/Threshold=-20dB

Una vez sobrepasado el umbral, el limitador se activa y se empieza a generar la función de ganancia estática que podemos ver en la primera gráfica de la figura 4.2. Como vimos que la mayor parte del segmento de audio sobrepasa el umbral del limitador entonces tenemos una función de ganancia estática con mucha atenuación. A continuación vemos la función de ganancia dinámica que implementa las constantes de tiempo de ataque y liberación. Como vemos la función dinámica es de cierta manera un filtro que tiene como salida la envolvente de la función estática, en el cual se calculan sus coeficientes con las constantes de tiempo del limitador.

La figura 4.3 es otro tipo de señal de audio procesada con el limitador. En este caso es se trata de la señal de audio de una grabación de batería. Vemos que la grabación es bastante alta en nivel puesto que abarca todo el rango de valores desde -1.0 hasta 1.0.

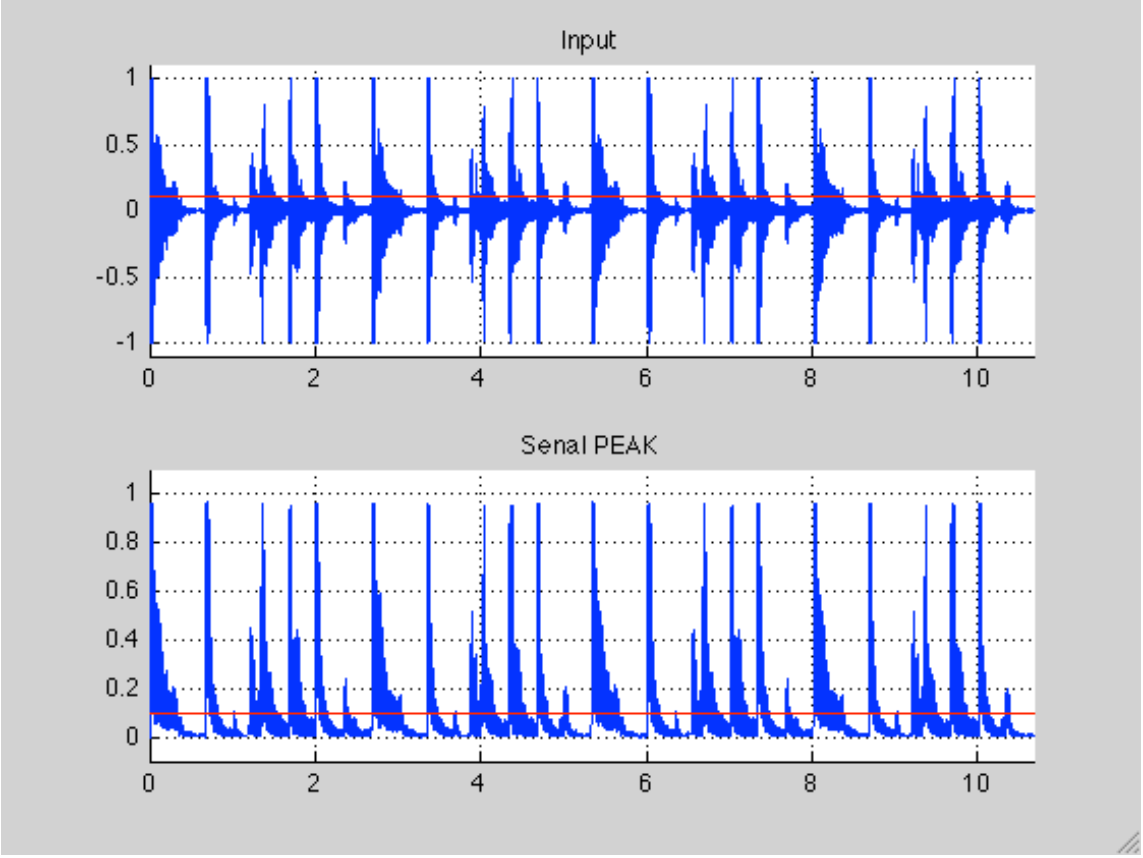


Figura 4.3. Señal de entrada y medición pico de nivel del limitador - Threshold=-20dB

En las figuras 4.4 y 4.5 podemos ver claramente los efectos en los cambios de los parámetros de ataque y liberación. Al ser la grabación una fuente percusiva, la señal contiene transitorios muy marcados. Por lo que si el limitador no actúa con suficiente rapidez, estos transitorios van a pasar y no van a ser atenuados, como se observa claramente en la figura 4.4. Para un ataque de 20ms los transitorios de más alto nivel no son suficientemente atenuados. Sin embargo al disminuir el tiempo de ataque, los transitorios son atenuados de manera correcta como se ve en la figura 4.5.

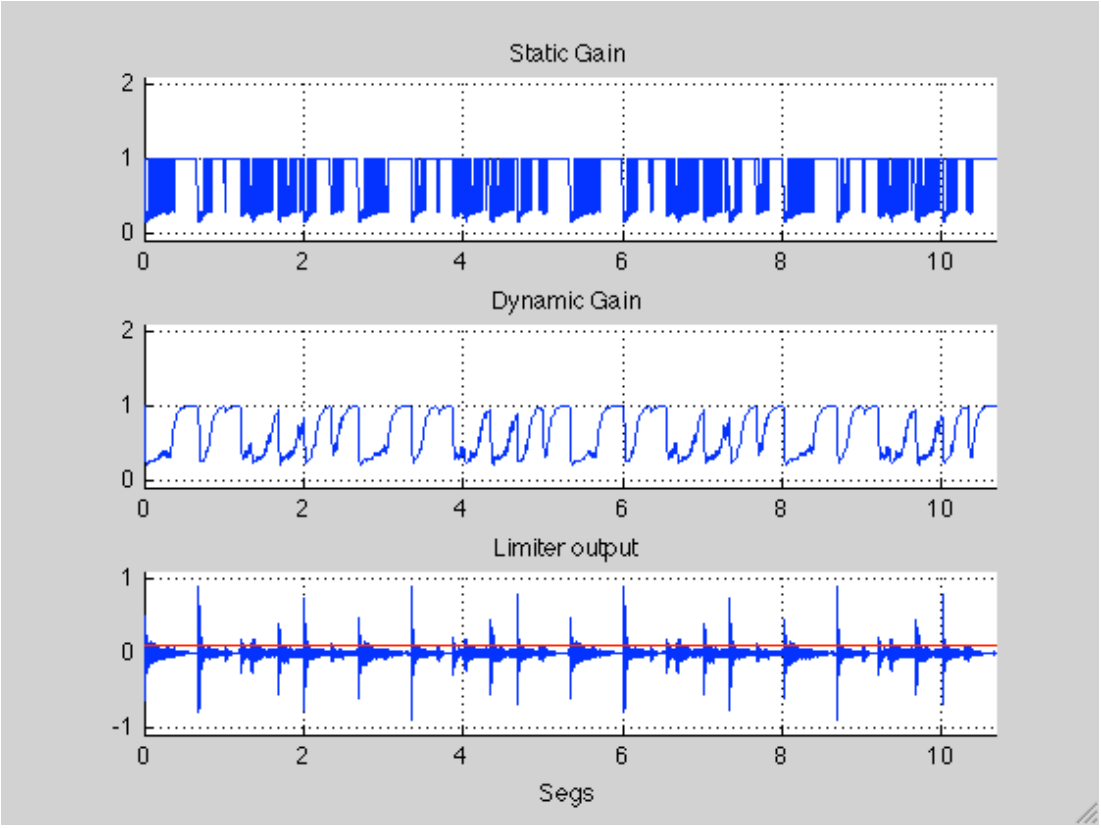


Figura 4.4. Limitador – Release=100ms/Attack=20ms/Threshold=-20dB

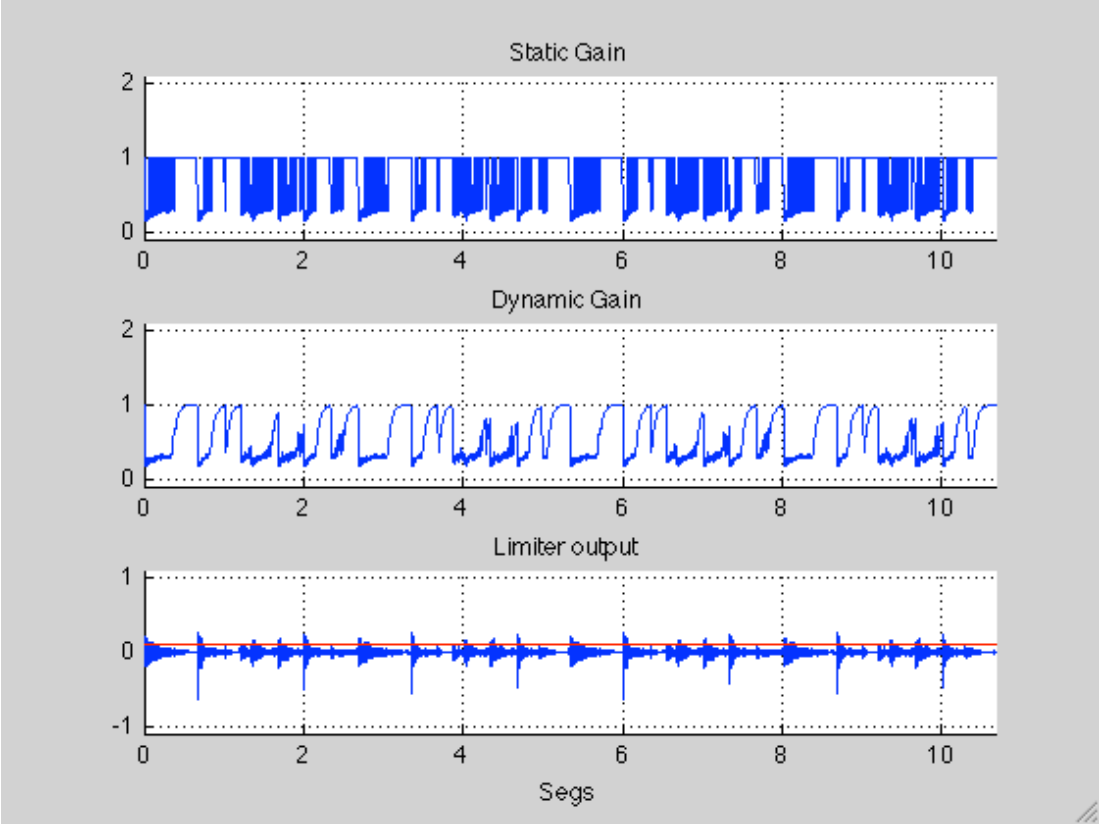


Figura 4.5. Limitador – Release=100ms/Attack=1ms/Threshold=-20dB

4.2 Compresor

En la figura 4.6 observamos la forma de onda de la entrada y su medición de nivel con el threshold. Esta señal corresponde a una mezcla musical completa, dado que una de las principales aplicaciones de un compresor es a mezclas musicales finales.

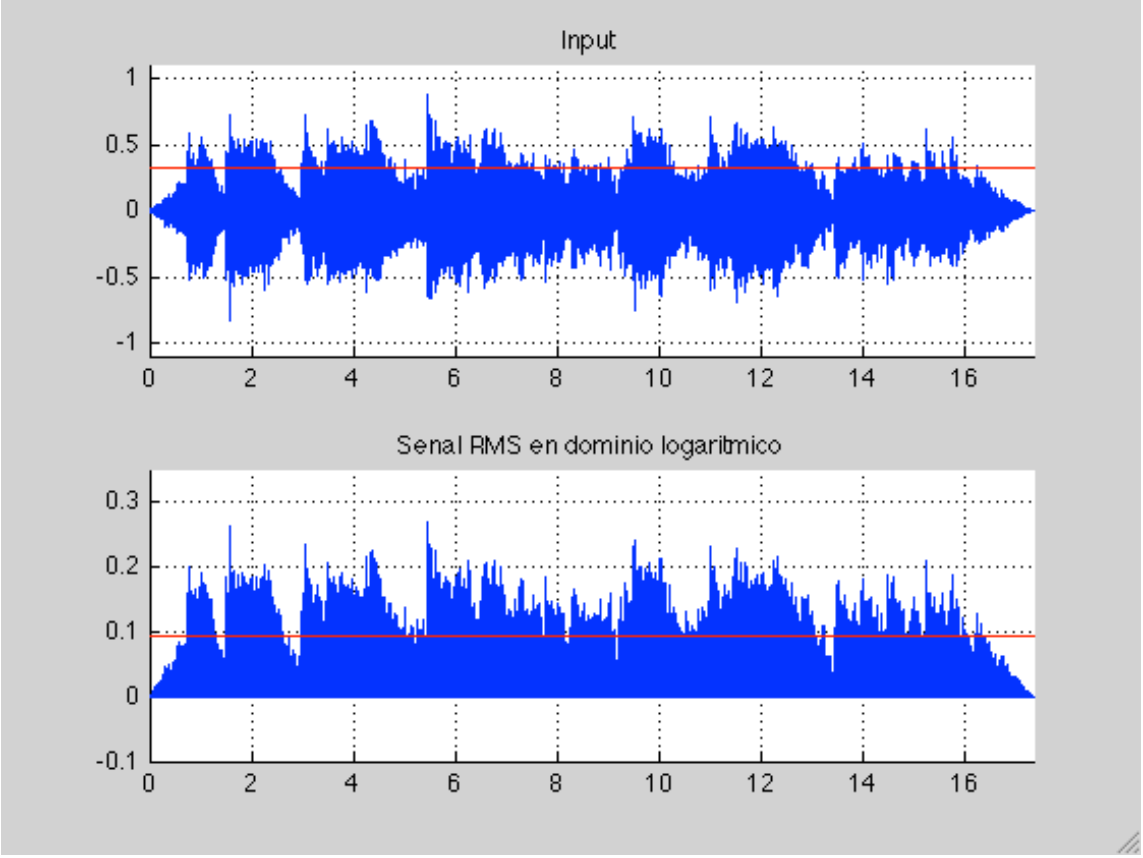


Figura 4.6. Señal de entrada y medición RMS de nivel del compresor - Threshold=-10dB

Los resultados en la figura 4.7 muestran que el compresor cumple con sus funciones, se ve claramente que la salida está "comprimida". La ganancia estática refleja el cálculo de la compresión cuando la señal de entrada sobrepasa el umbral y vemos como los parámetros de tiempo funcionan bien con los valores de 100ms para la liberación y 20ms para el ataque.

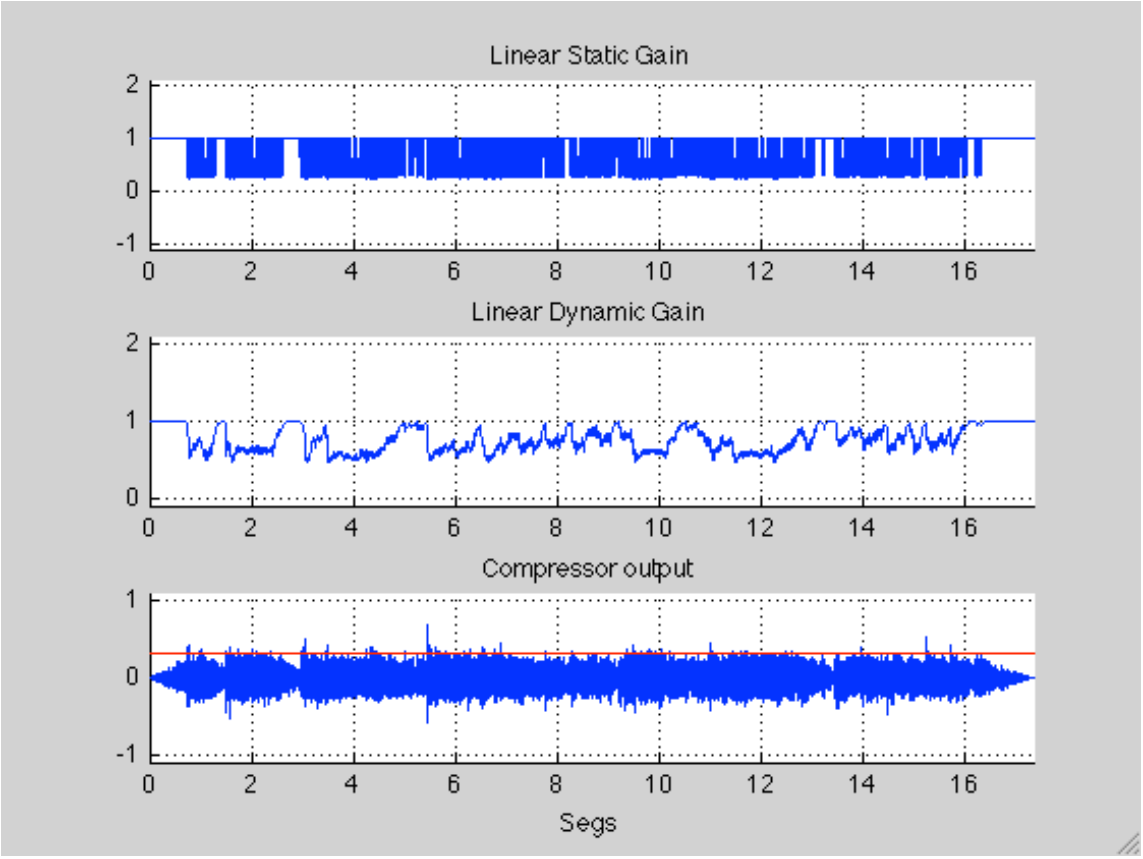


Figura 4.7. Compresor – Release=100ms/Attack=20ms/Ratio=5:1/Threshold=-10dB

Sin embargo para mostrar un poco más claramente los efectos de estos parámetros de tiempo, en la figura 4.8 se exagera el valor del tiempo de liberación lo que da como resultado que una vez que la señal de entrada sobrepasa el umbral la atenuación se efectúa pero como el tiempo de liberación es tan largo el compresor está prácticamente activo todo el tiempo atenuando mucho más la señal.

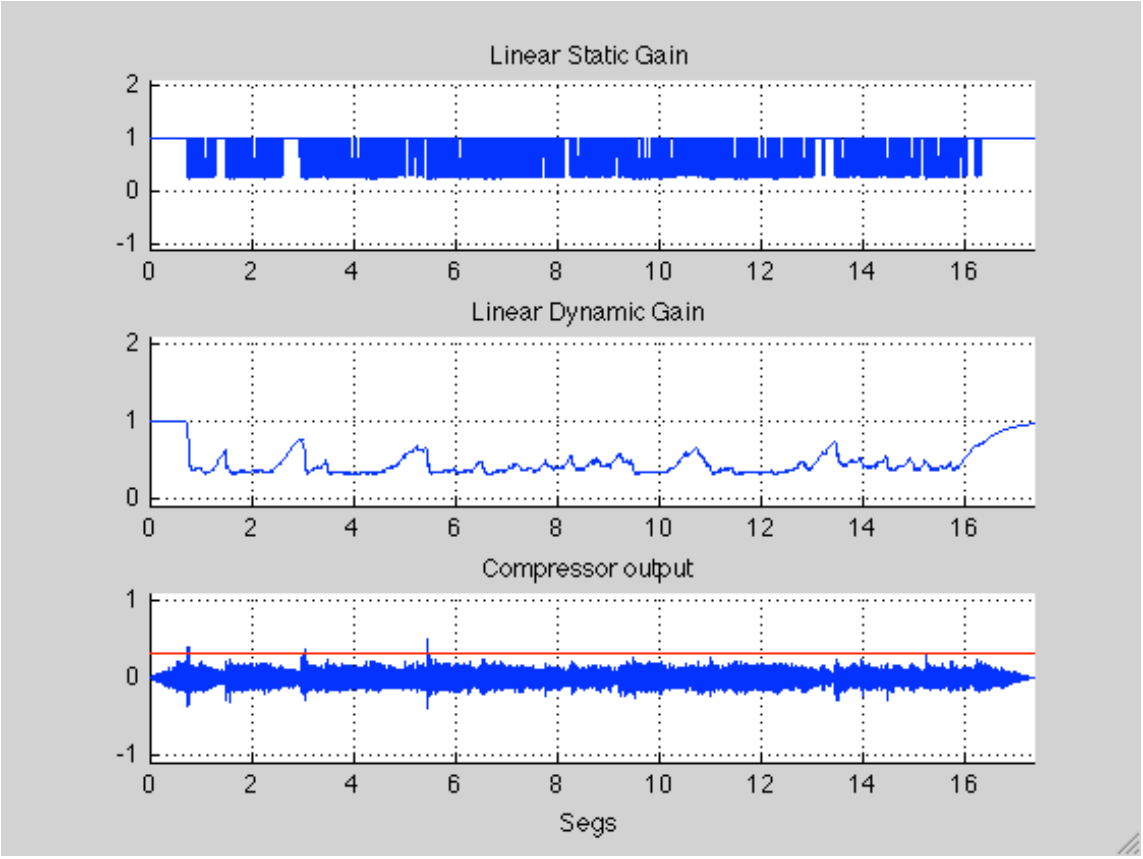


Figura 4.8. Compresor – Release=1s/Attack=20ms/Ratio=5:1/Threshold=-10dB

Por otro lado un tiempo de ataque muy lento, no puede detectar los transitorios de alto nivel y el compresor no puede realizar bien su trabajo como lo vemos en la figura 4.9 donde la ganancia dinámica es bastante inferior a la mostrada en las figuras anteriores. Es importante recalcar que en estos casos hemos variado solo los parámetros de tiempo sin tocar el threshold ni la relación entrada/salida. De esta manera vemos que estos parámetros también influencian en la cantidad de atenuación que será aplicada a la señal.

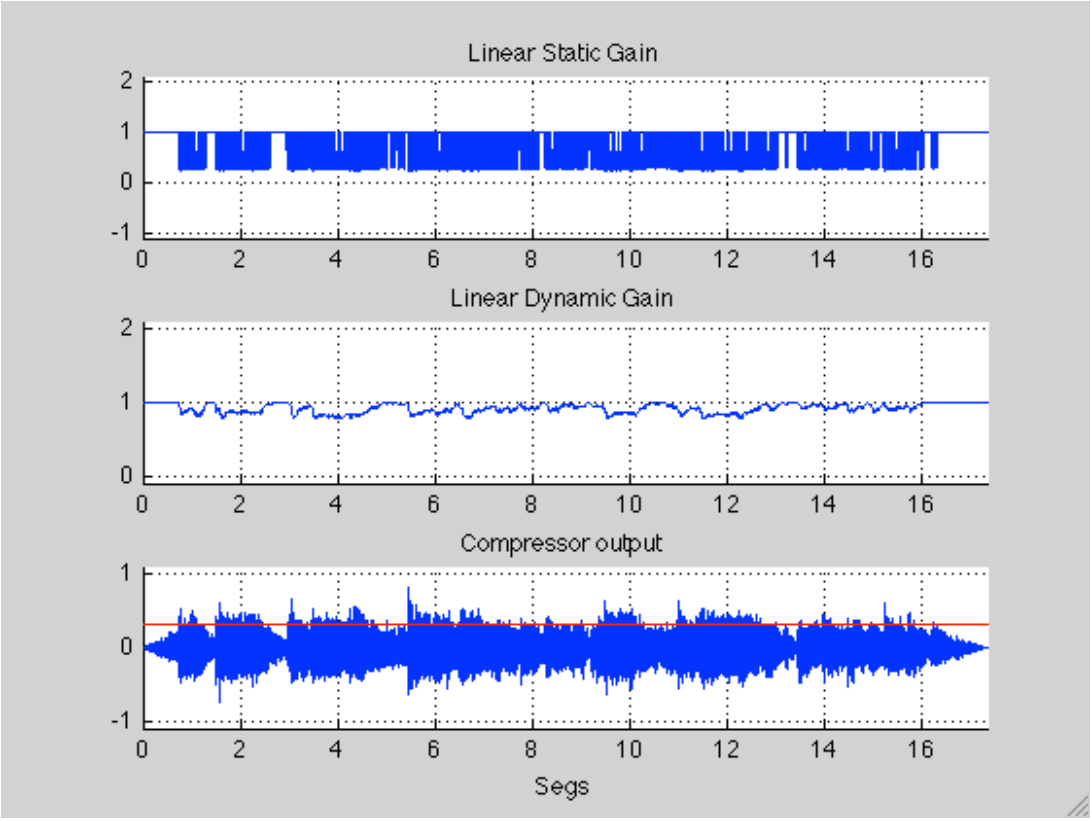


Figura 4.9. Compresor – Release=100ms/Attack=100ms/Ratio=5:1/Threshold=-10dB

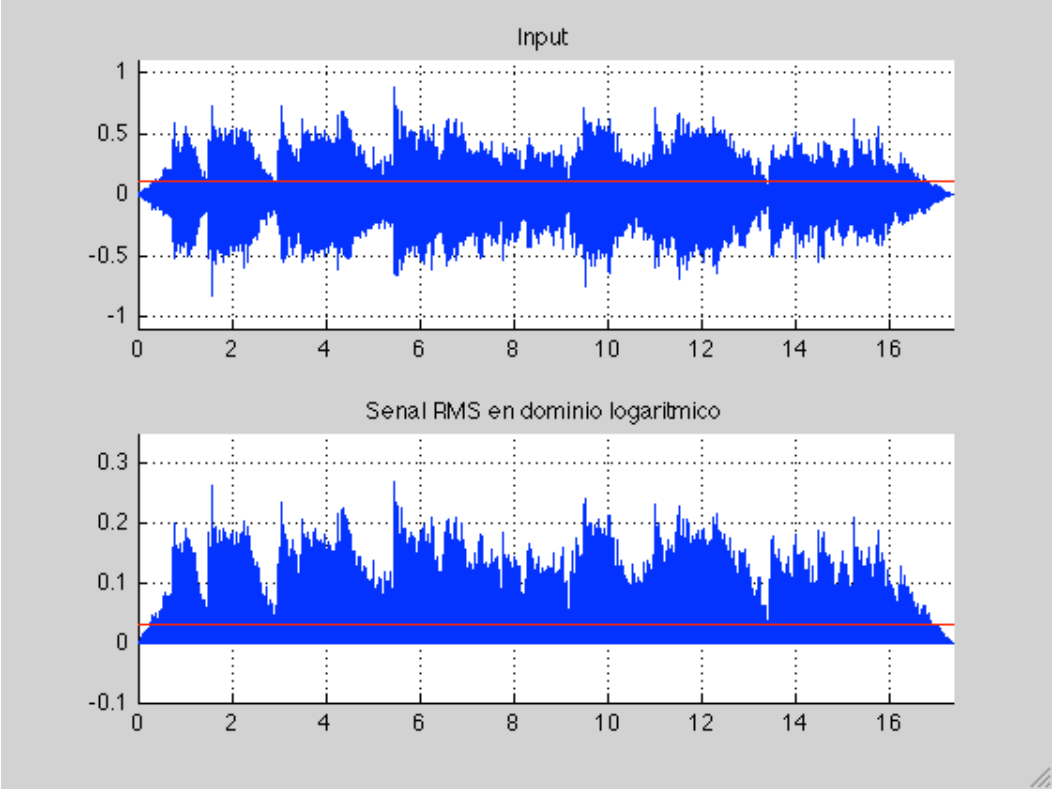


Figura 4.10. Señal de entrada y medición RMS de nivel del compresor - Threshold=-20dB

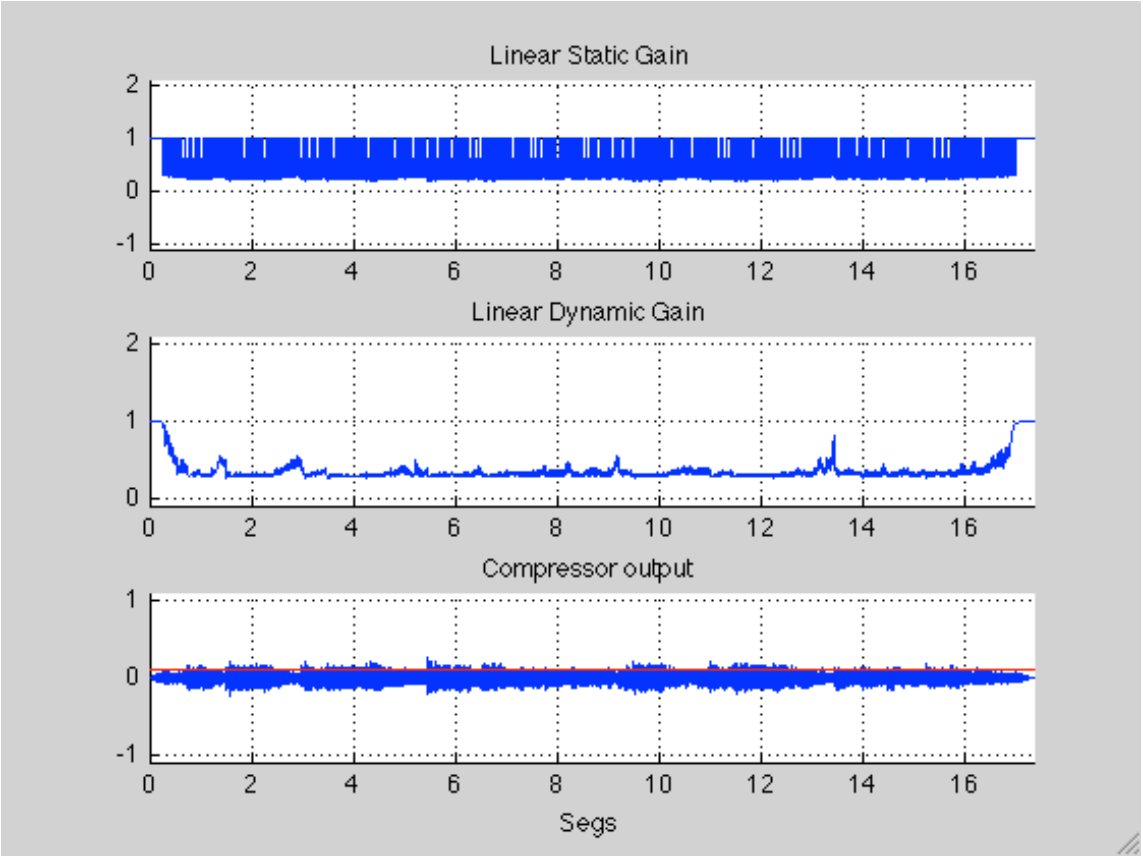


Figura 4.11. Compresor – Release=100ms/Attack=10ms/Ratio=10:1/Threshold=-20dB

El interés de las siguientes figuras es de mostrar el efecto de los parámetros de threshold y ratio. La figuras 4.12 y 4.14 corresponden a una mezcla musical completa y a una grabación de voz respectivamente. Vemos como al incrementar el ratio y hacer que el nivel del umbral sea más bajo, se realiza una compresión mucho más importante, resultando en un nivel de salida mucho más bajo.

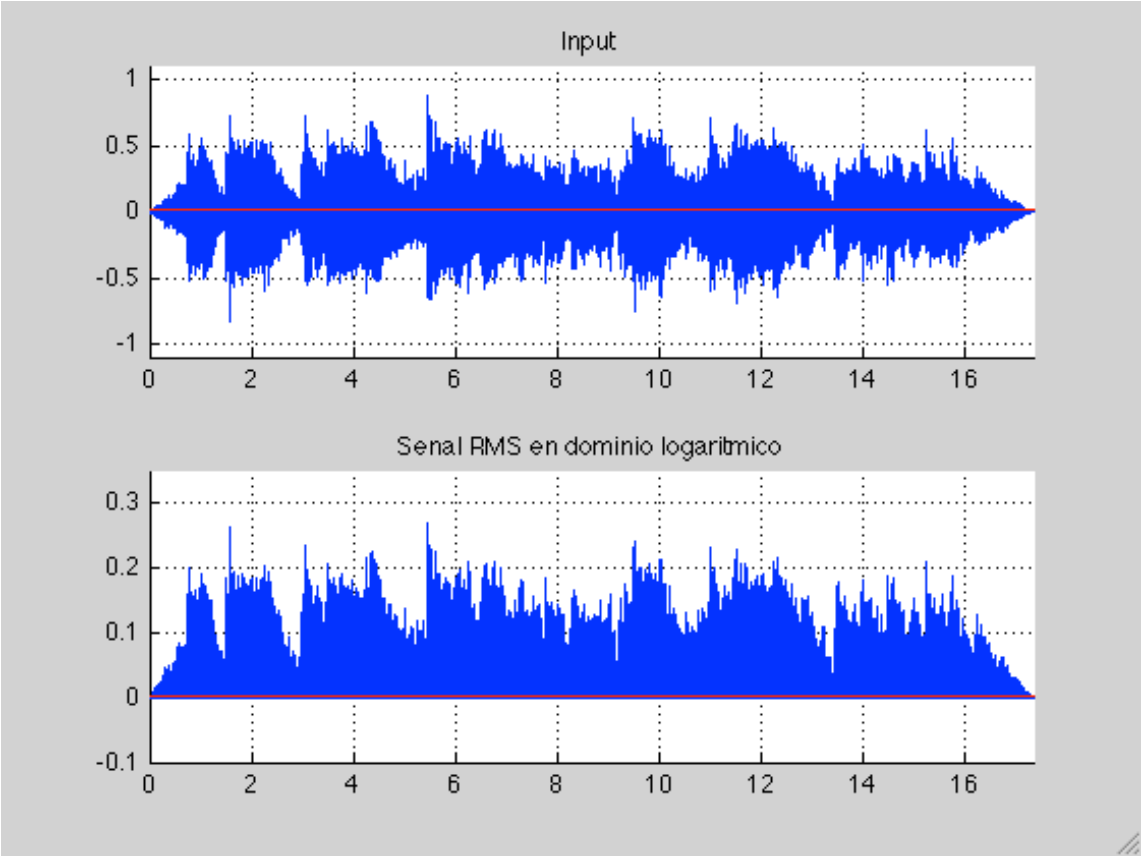


Figura 4.12. Señal de entrada y medición RMS de nivel del compresor - Threshold=-40dB

En la figura 4.13, que es una grabación de voz, vemos claramente como al incrementar los parámetros de threshold y ratio la señal se comprime de manera considerable. Para la compresión de mezclas finales por lo general no se tocan mucho los valores de los parámetros de tiempo sino que se juega más con los valores del threshold y del ratio para conseguir los resultados deseados.

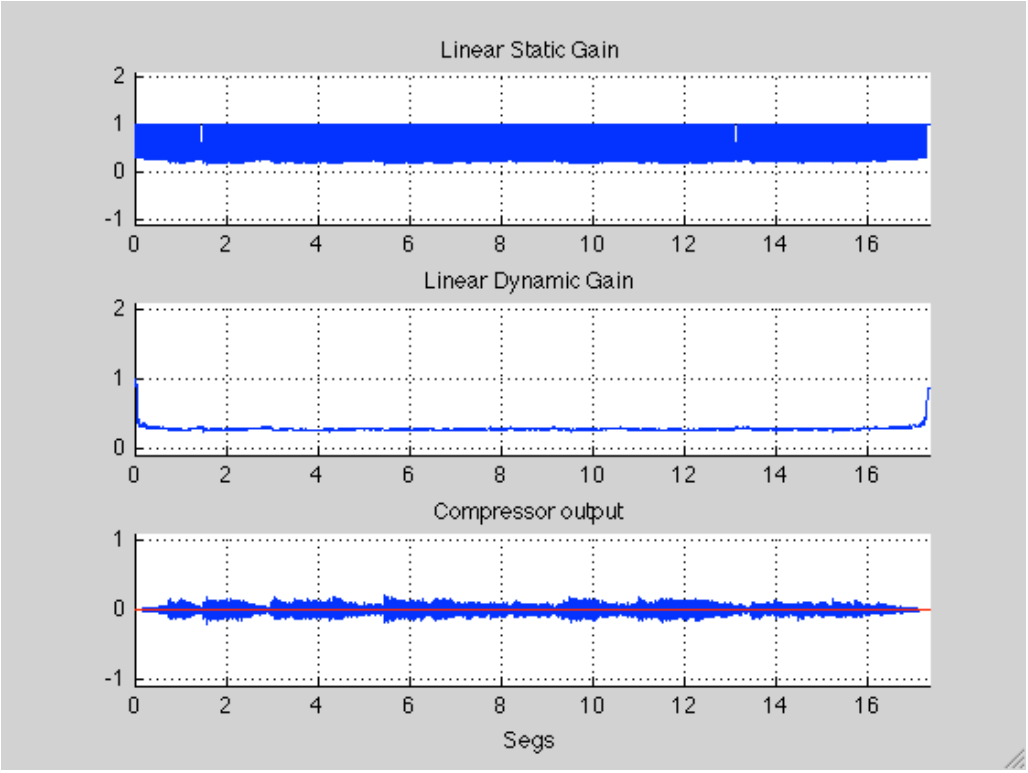


Figura 4.13. Compresor – Release=100ms/Attack=10ms/Ratio=20:1/Threshold=-40dB

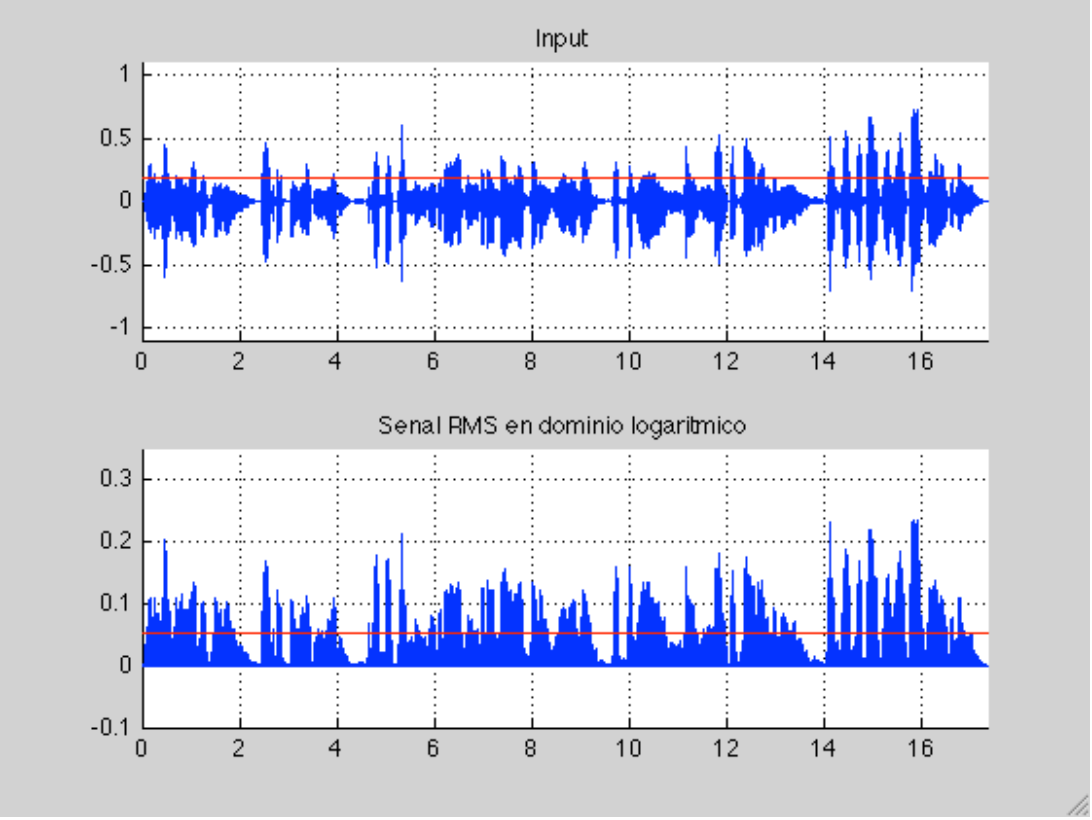


Figura 4.14. Señal de entrada y medición RMS de nivel del compresor - Threshold=-15dB

Por otro lado la señal de una grabación de voz (figura 4.14) debe ser tratada de una manera un poco más delicada, es decir que todos los parámetros juegan un papel importante en el resultado final. Con los parámetros de tiempo se llega a eliminar cualquier transitorio que por lo general se traduce como susurros que perturban al oyente, y con los parámetros de threshold y ratio se logra una cierta uniformidad en la señal, lo que se traduce como un sonido más agradable al oído. En las figuras 4.15 y 4.16 vemos como variando el tiempo de ataque podemos obtener resultados muy diferentes, ya que un ataque rápido elimina los transitorios.

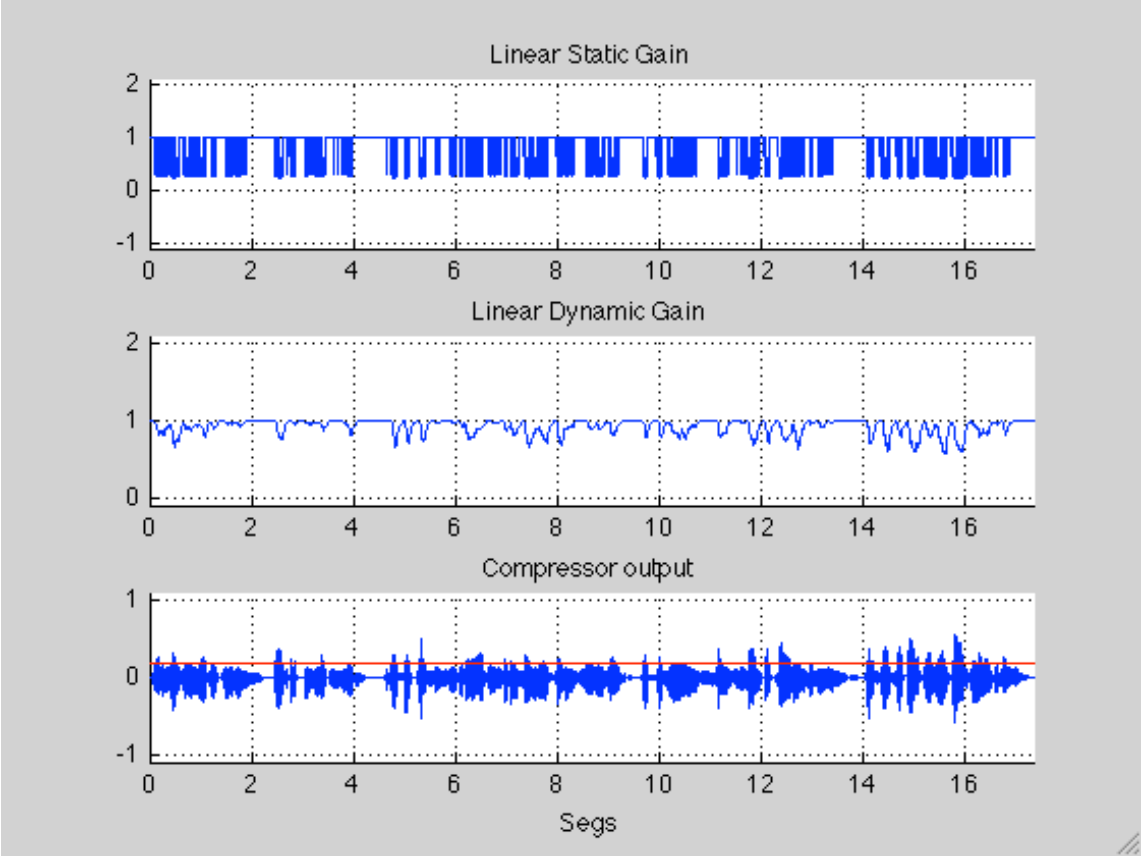


Figura 4.15. Compresor – Release=100ms/Attack=100ms/Ratio=5:1/Threshold=-15dB

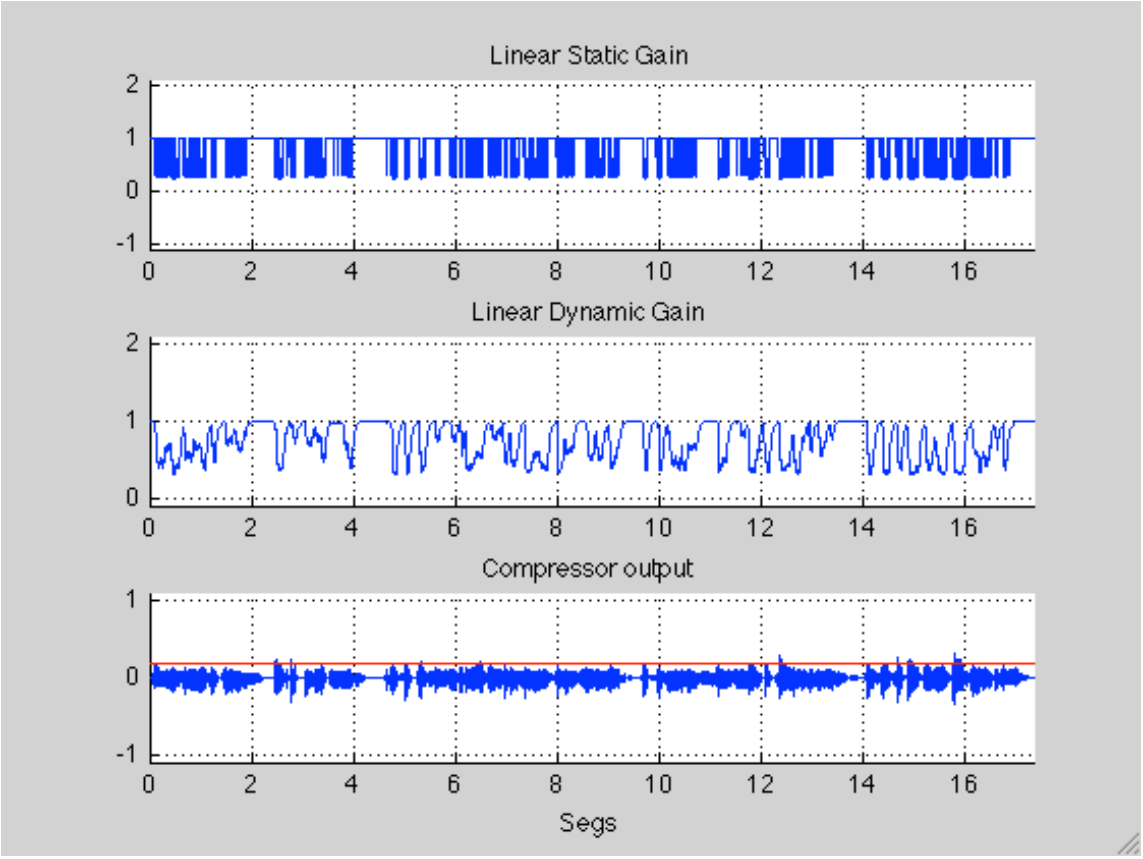


Figura 4.16. Compresor – Release=100ms/Attack=10ms/Ratio=5:1/Threshold=-15dB

Las figuras 4.17 y 4.18 muestran la respuesta impulsional del compresor. En las gráficas vemos la entrada impulsiva y la respuesta de la ganancia dinámica. Podemos verificar que el tiempo de ataque (tiempo en que la señal pasa del 10% al 90% de su valor máximo) y el tiempo de liberación (tiempo en que la señal pasa del 90% al 10% de su valor máximo) se aplican efectivamente en el compresor. El principio de implementación de las variables de tiempo es el mismo para todos los tratamientos dinámicos, por eso estas figuras son también válidas para los otros tratamientos.

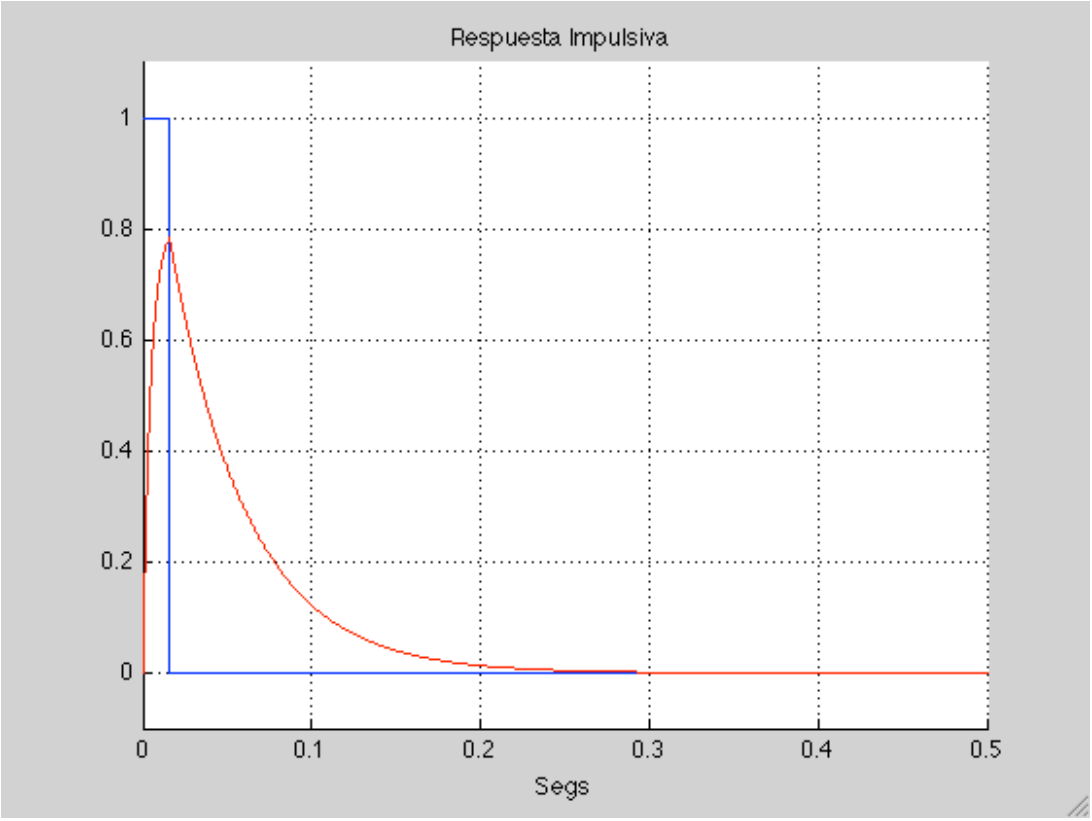


Figura 4.17. Compresor – Release=100ms/Attack=10ms

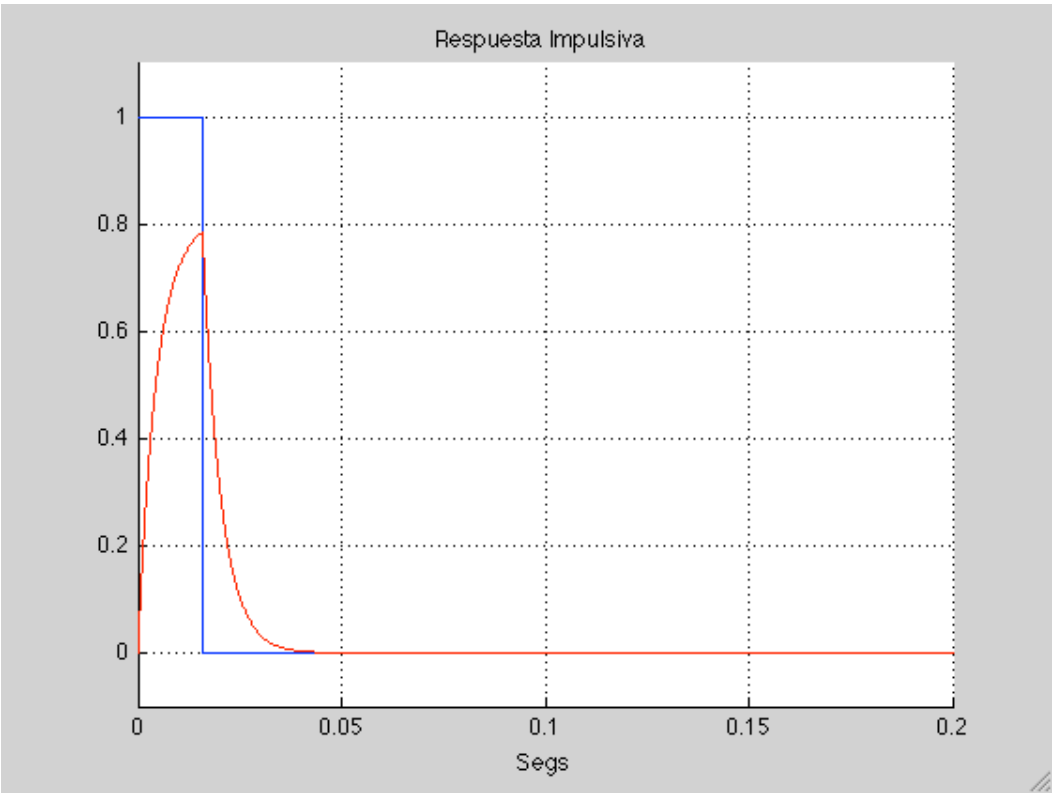


Figura 4.18. Compresor – Release=10ms/Attack=10ms

En las figuras 4.19 y 4.20 vemos la configuración alternativa de un compresor que se conoce como “Voice over compression”. Tenemos en primer lugar la medición de la señal que va a comprimir la señal que entra y sale del compresor. Finalmente vemos en la figura 4.20 como las ganancias estática y dinámica varían en función de la señal que va a comprimir. La etapa final es multiplicar esta ganancia por la entrada del compresor y como resultado tenemos una señal comprimida por otra. Esto se evidencia con claridad en la última gráfica donde en rojo tenemos el nivel RMS (invertido para hacer más evidente el resultado) que comprime la señal de salida. Cuando el nivel de la señal que comprime es alto, la atenuación de la señal de salida es mayor.

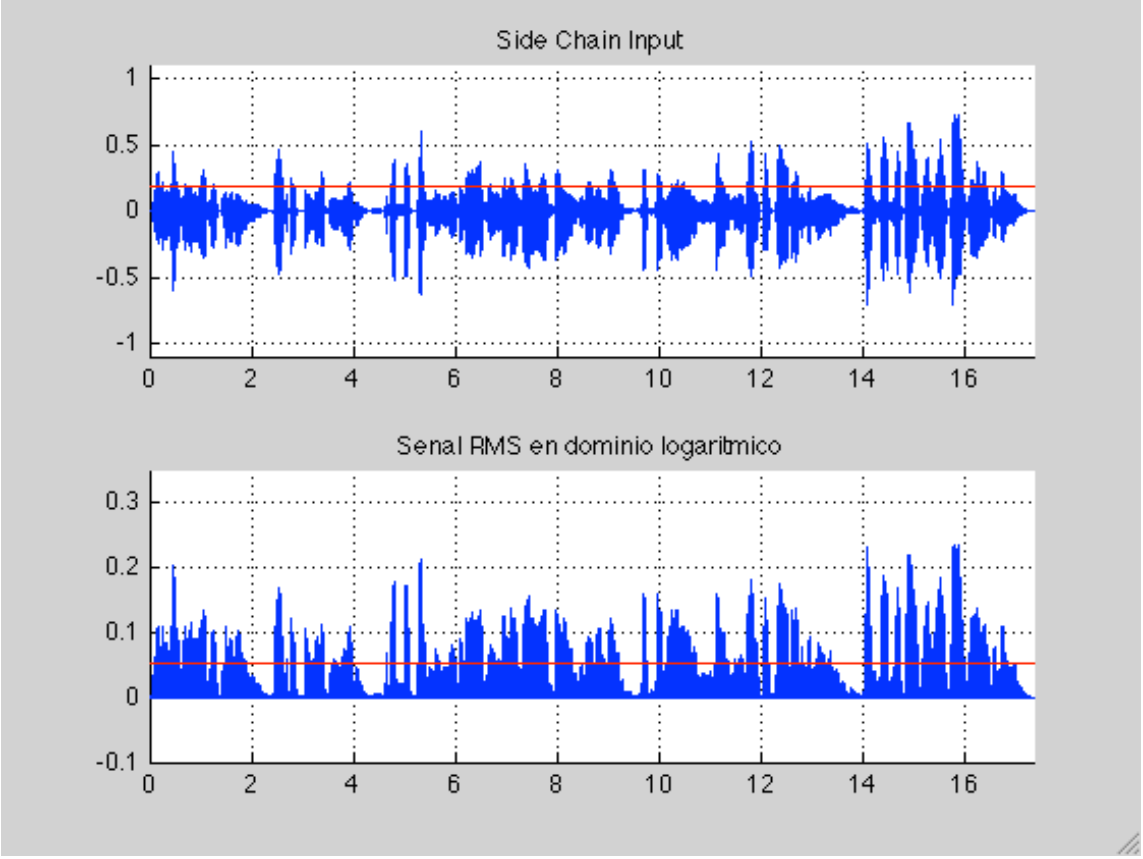


Figura 4.19. Señal de entrada y medición RMS de nivel del side chain - Threshold=-15dB

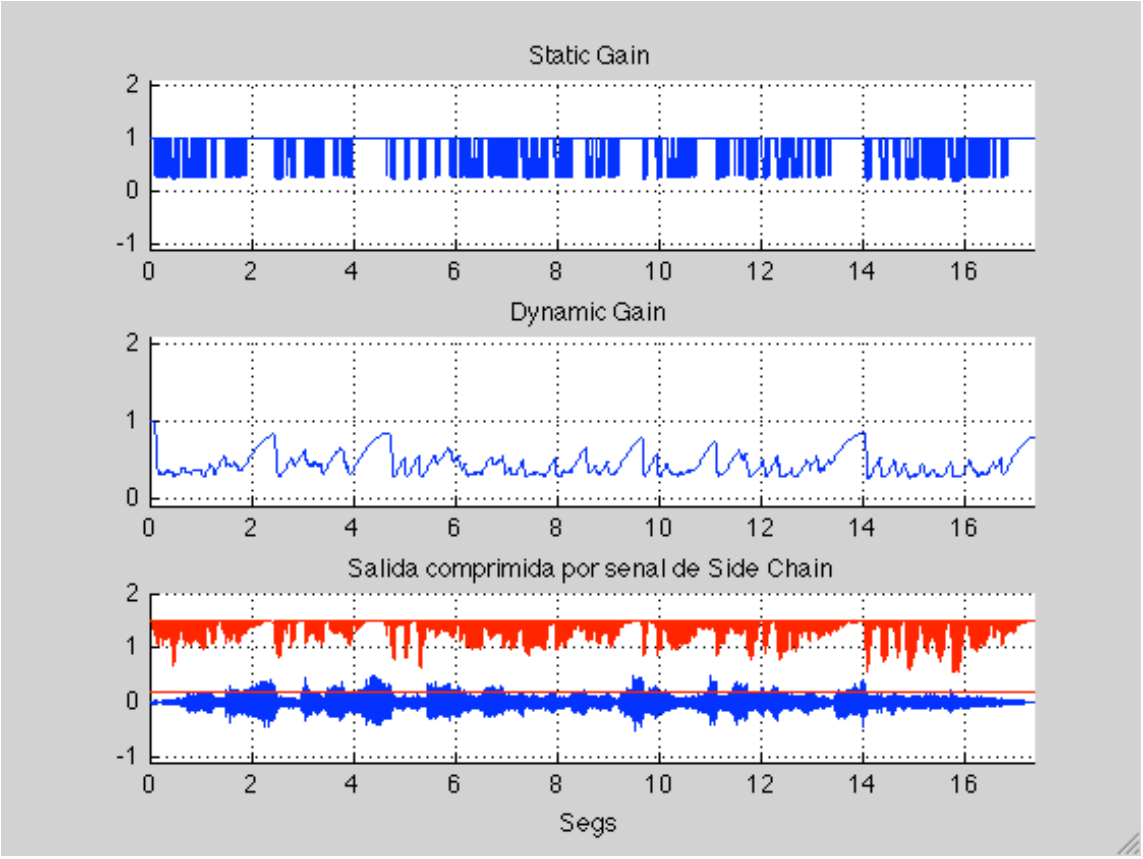


Figura 4.20. Compresor – Release=1s/Attack=10ms/Ratio=5:1/Threshold=-15dB

4.3 Expansor

El expansor es probablemente de los cuatro procesamientos el menos usado. Para mostrar su funcionamiento usamos una grabación de batería. Recordemos que el expansor atenúa los niveles por abajo del threshold.

La figura 4.21 nos muestra la señal de entrada y su medición de nivel RMS.

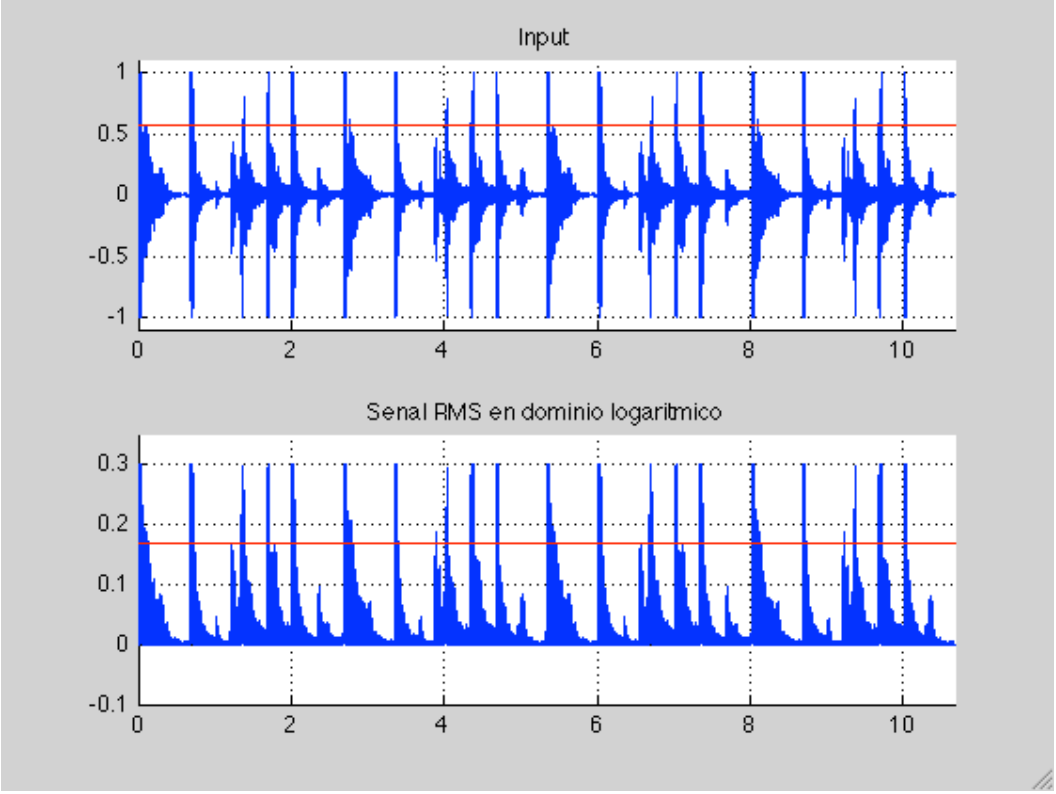


Figura 4.21. Señal de entrada y medición RMS de nivel del expansor - Threshold=-5dB

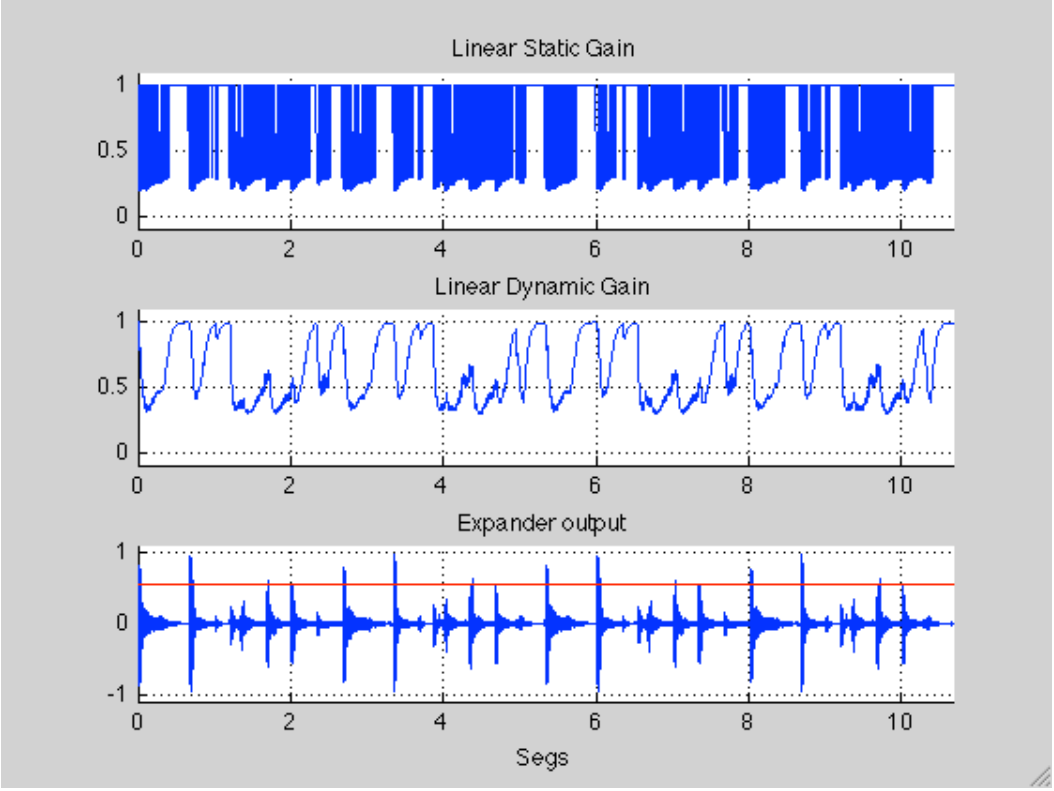


Figura 4.22. Expansor – Release=100ms/Attack=50ms/Ratio=10:1/Threshold=-5dB

En las figuras 4.22 y 4.23 vemos como variando los parámetros de tiempo y ratio los niveles bajos son atenuados. Sin embargo también podemos ver en la figura 4.22 también se atenúan un poco los niveles altos. Esto se corrige variando los parámetros de tiempo. En efecto, en la figura 4.23 vemos que los niveles altos están intactos.

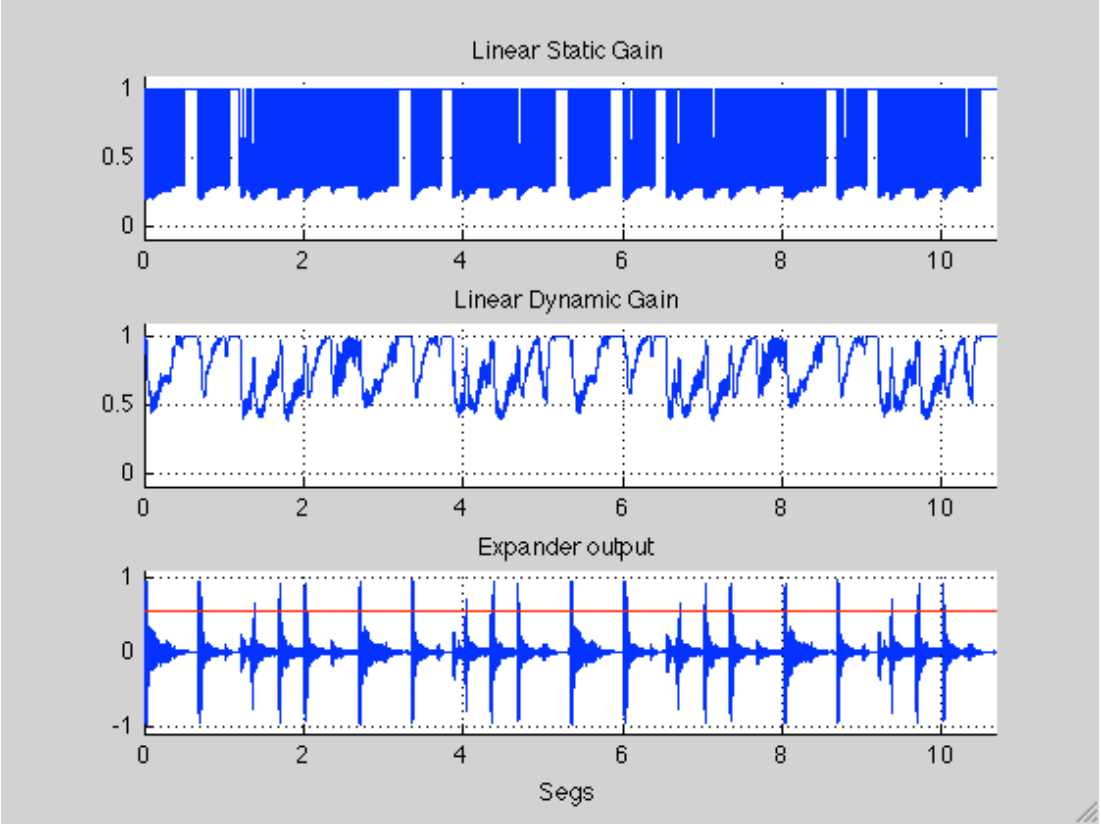


Figura 4.23. Expansor– Release=10ms/Attack=50ms/Ratio=20:1/Threshold=-5dB

4.4 Compuerta de ruido

La compuerta de ruido es un efecto que se usa mucho en grabación de baterías por lo tanto para las pruebas del compresor usamos dos grabaciones de batería. La idea es mostrar como para los dos casos, variando los parámetros de la compuerta de ruido se obtienen diferentes resultados, que para el caso de la compuerta de ruido se pueden oír claramente.

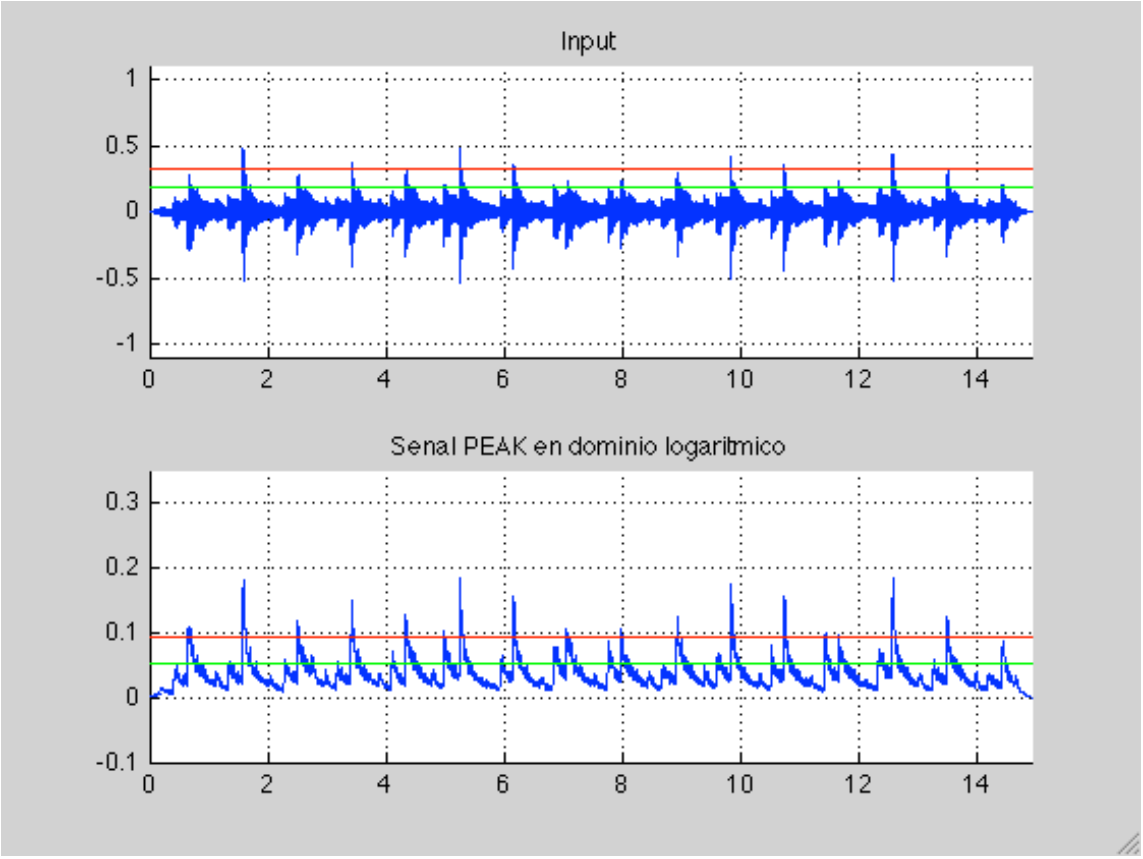


Figura 4.24. Señal de entrada y medición pico de nivel de la compuerta de ruido - Threshold=-20dB

Se deben desde luego adecuar los valores de todos los parámetros para que estos tengan efecto sobre la señal, principalmente el umbral y histéresis ya que son estos dos parámetros los que activan la apertura y clausura de la compuerta.

En la figura 4.24 tenemos la medición de nivel pico de una grabación del bombo de una batería. Como podemos ver, en la figura 4.25, para la compuerta de ruido no existe una gráfica de ganancia estática puesto que la compuerta de ruido no realiza ningún cálculo de ganancia sino que simplemente abre o cierra la compuerta, dándole a la ganancia final valores de 0 o 1. Desde luego existen tiempos de transición entre los estados de apertura y clausura de la compuerta. Los tiempos de estos estados de transición están determinados por las contantes de tiempo de ataque, liberación y retención.

La figura 4.25 muestra la señal de salida donde únicamente queda el sonido del bombo. Todo el sonido "parásito" del resto de elementos de la batería que también es grabado por ese micrófono es completamente eliminado. Sin embargo es importante no realizar la eliminación sistemática del resto de la señal ya que componentes como por ejemplo la reverberación y el tiempo de decaimiento de la señal también pueden ser modificados y así se distorsiona el sonido original de la fuente grabada.

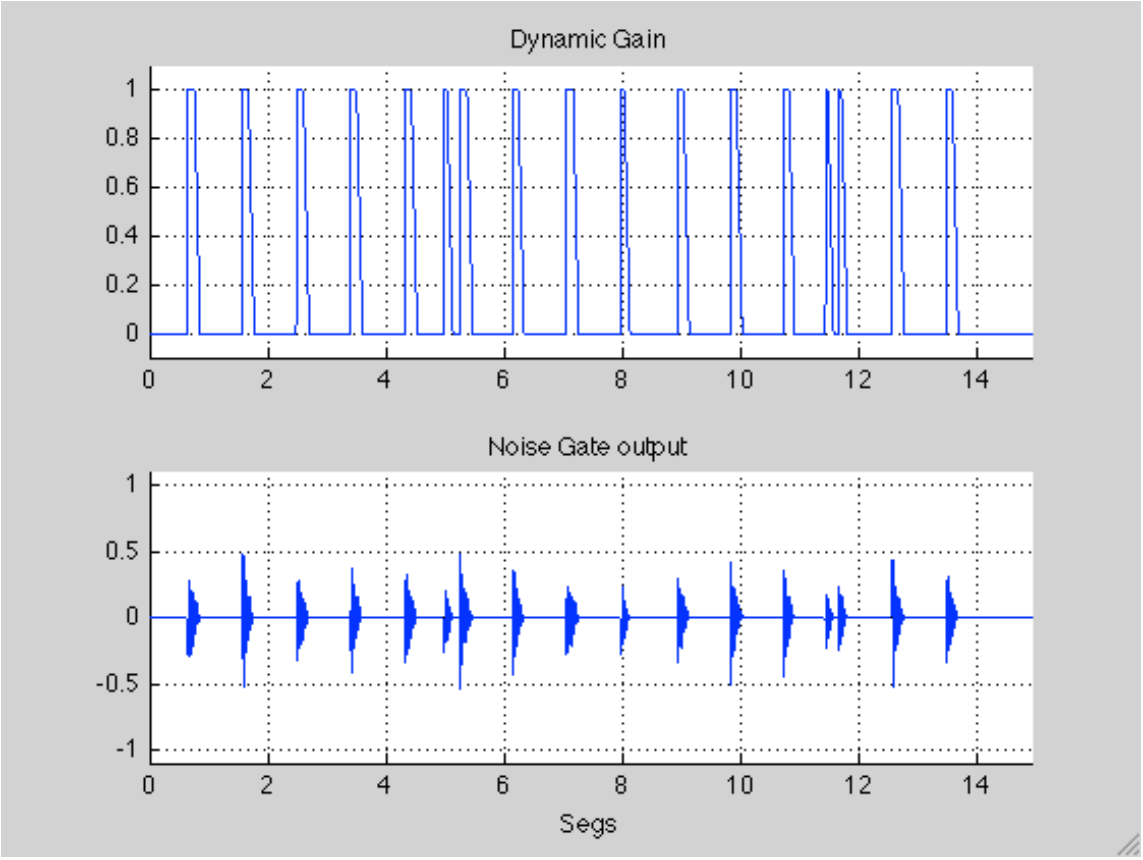


Figura 4.25. Compuerta de ruido-Release=100ms/Attack=0.3ms/Hold=5ms/Threshold=-10dB/Hysteresis=5dB

Las figuras 4.26 y 4.27 muestran los efectos menos drásticos de una compuerta de ruido sobre una grabación de baterías. El threshold e histéresis tiene valores que atenúan niveles realmente bajos por lo que la mayoría de la señal queda prácticamente intacta a diferencia de la señal de salida de la figura 4.25 donde se elimina más del 60% del contenido de la señal.

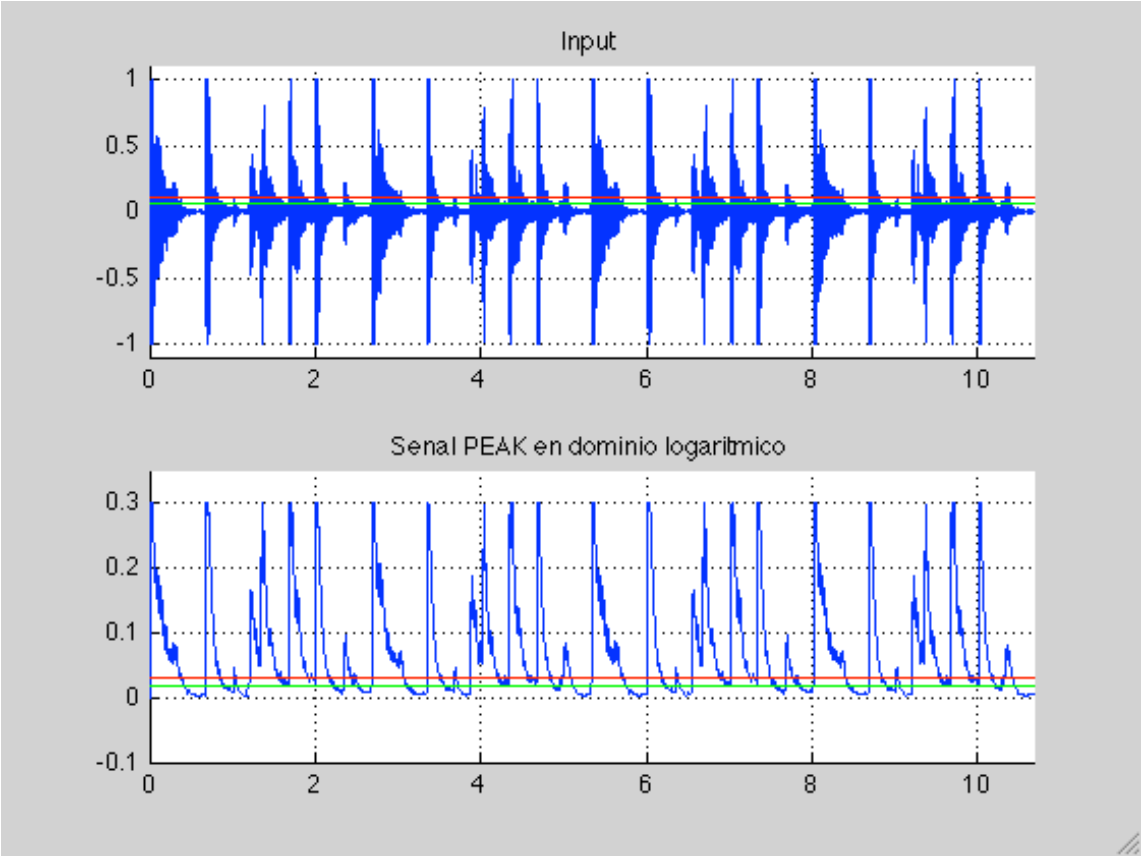


Figura 4.26. Señal de entrada y medición pico de nivel de la compuerta de ruido - Threshold=-20dB

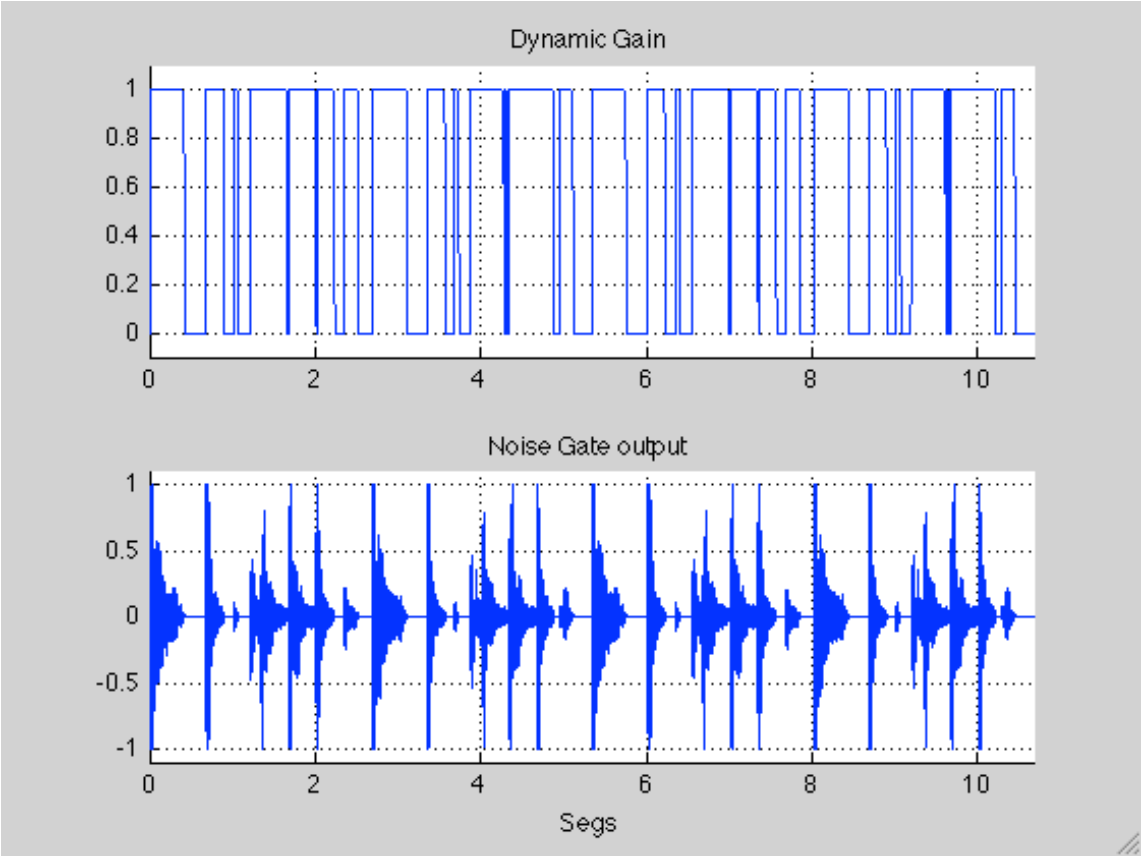


Figura 4.27. Compuerta de ruido-Release=10ms/Attack=0.1ms/Hold=10ms/Threshold=-20dB/Hysteresis=4dB

Capítulo 5 Implementación

Esta sección provee una breve una descripción general de un plug-in de audio, seguido por una lista de diferentes tipos de plug-ins de uso común. A continuación se detalla la implementación de los algoritmos de control de rango dinámico. La información contenida en este capítulo constituye una parte muy importante de esta tesis y por lo tanto requiere un conocimiento previo de la sintaxis y terminología de C++. Este capítulo está redactado de tal manera que un lector interesado sea capaz de entender la estructura global del código fuente de un plug-in VST, aún sin conocimiento extenso de C++.

Para la implementación el framework VSTSDK 2.4 para C++ fue utilizado. El código fuente se compiló utilizando Xcode1 de MacOS. El código fuente de un plug-in VST es idéntico para todas las plataformas compatibles.

5.1 Que es un plug-in?

Un plug-in de audio es un componente de software que no puede ser ejecutado en un computador por sí solo, sino que necesita una aplicación anfitriona que haga uso del procesamiento de la señal de audio que el plug-in provee. De esta manera la aplicación anfitriona provee las funciones básicas como entradas y salidas de audio, a través de una interfaz de audio, lectura y escritura de archivos de audio desde y hacia el disco duro, y edición de la forma de onda. Al ubicar el archivo que contiene el código del plug-in compilado en un directorio al cual la aplicación anfitriona tiene acceso, el plug-in se hace disponible para su uso dentro de la aplicación anfitriona[12]. En otras palabras, el plug-in aumenta la funcionalidad de la aplicación anfitriona al proveer

1 http://developer.apple.com/technologies/xcode.html

procesamiento especializado de la señal de audio. El anfitrión no necesita saber nada acerca del algoritmo DSP dentro del plug-in, y el plug-in no necesita ningún conocimiento del manejo de entrada y salida o de la interfaz gráfica de usuario del anfitrión. La siguiente es una cita de las especificaciones VST:

"Desde el punto de vista de la aplicación anfitriona, un plug-in VST es una caja negra con un número arbitrario de entradas, salidas y parámetros asociados. El anfitrión no necesita ningún conocimiento del proceso del plug-in para poder utilizarlo."[3]

Entonces, la única cosa en la cual la aplicación anfitriona y el plug-in tienen que estar de acuerdo es la manera en que comunican entre ellos las muestras de audio y los parámetros de valores. El anfitrión provee continuamente al plugin porciones de muestras de audio, buffers de entrada, los cuales son procesados por el plugin utilizando su algoritmo DSP y luego son retornados al anfitrión como buffers de salida para su reproducción.

5.2 Tipos de plug-ins

La manera mediante la cual un plug-in y un anfitrión se comunican varía entre los diferentes tipos de plug-in y sus especificaciones. Existen varios formatos diferentes en el mercado. El tema de esta tesis es el formato de plug-in presentado en 1996 por Steinberg Media Technologies AS, en su línea de productos VST (Virtual Studio Technology). El Kit de Desarrollo de Software de plug-ins VST (VSTSDK 2.4 – Virtual Studio Technology Plug-in Specification 2.4 Software Development Kit) se puede descargar gratuitamente de la página web de la compañía. El VSTSDK 2.4 consiste de un Framework C++, y se provee un código fuente para MacOS y Windows. Esto significa que los plug-ins VST son fácilmente desarrollados independientemente de la plataforma y son compatibles con distintas plataformas [12].

La disponibilidad gratuita del VSTSDK 2.4 ha generado un sin número de plug-ins desarrollados por terceras partes, y el formato de plug-in VST es uno de los más grandes formatos compatibles con aplicaciones anfitrionas en la actualidad. Otros formatos de plug-ins comunes son:

- Direct X desarrollado por Microsoft es realmente más que un formato de plug-in. Es una agrupación de interfaces de programación de aplicaciones (API) para el desarrollo de aplicaciones multimedia. Los plug-in Direct X solo trabajan en aplicaciones anfitrionas de Windows. Un SDK C++ para escribir aplicaciones de Direct X, incluyendo plug-ins, está disponible en el website de Microsoft.2

- MAS es un formato de plug-in desarrollado por Mark Of The Unicorn (MOTU). Las siglas MAS corresponden a Sistema de Audio MOTU en inglés. Estos plug-ins pueden ser utilizados solo en MacOS. Al firmar un acuerdo con MOTU, los desarrolladores independientes pueden obtener el SDK de MOTU.3

- TDM es un formato de plug-in desarrollado por Digidesign. Las siglas TDM significan Acceso Multiple por Divisón de Tiempo (Time Division Multiplexing) en inglés. Este tipo de plug-ins requieren la presencia de un chip DSP dedicado, y por lo tanto difieren de los otros tipos de plug-ins listados aquí, los cuales utilizan el CPU de la computadora en lo que se conoce como procesamiento nativo o basado en anfitrión. Para desarrollar plug-ins TDM los desarrolladores independientes deben firmar un acuerdo con Digidesign.

- AS/RTAS son formatos de plug-in desarrollados por Digidesign para procesamiento nativo. AS (Audio Suite) significa “Espacio de Audio” en inglés

2 http://www.microsoft.com/downloads/es-es/default.aspx

3 http://www.motu.com/

y RTAS (Real Time Audio Suite) significa “Espacio de Audio en Tiempo Real” en inglés. RTAS permite que el resultado del procesamiento del plug-in sea escuchado mientras los cálculos son realizados, en contraste con los plug-ins AS con los cuales todo el archivo o sección de audio debe ser procesado antes de que este pueda ser escuchado. Para desarrollar plug-ins AS y RTAS, los desarrolladores independientes deben firmar un acuerdo con Digidesign.

- LADSPA es un formato de plug-in para el sistema operativo Linux. Sus siglas corresponden a “Linux Audio Developers’ Simple Plug-in API”, y se distribuyen bajo la licencia pública “Less-GNU Public License”. Un SDK C/C++ se puede descargar del Internet.4

- Audio Units (Unidades de Audio) desarrollado por Apple es un formato plug-in que es parte del MacOS X Core Audio. Un SDK y herramientas de programador están disponibles en el website de Apple. Emagic ofrece una biblioteca para facilitar la conversión de plug-ins VST a plug-ins AU.5

5.3 Plug-ins de procesamiento en tiempo real y no real

Arfib [2] describe un procesamiento de audio DSP en tiempo real como un procesamiento que puede ser escuchado al mismo tiempo que se realizan los cálculos. Para que esto sea posible el tiempo promedio de procesamiento de una muestra de una señal mono debe ser menor al periodo de muestreo de la señal digital de audio. Por otro lado los plug-ins que no tratan la señal en tiempo real procesan toda la sección de audio antes de que esta pueda ser escuchada, por lo que tienen restricciones de tiempo más

4 http://www.ladspa.org/ladspa_sdk

5 http://developer.apple.com/audio

flexibles. Normalmente un segmento corto del archivo de audio puede ser previsualizado para realizar ajustes a los parámetros del plug-in. Cuando los parámetros se definen, el plug-in aplica su proceso a todas las muestras del archivo, y escribe los resultados en un archivo de audio nuevo, el cual puede ser escuchado posteriormente.

El poder de procesamiento de las computadoras personales actuales puede manejar con facilidad los plug-ins de tiempo real. Utilizando la especificación VST, los plug-ins se pueden desarrollar en tiempo real y fuera de tiempo real. El plug-in desarrollado en esta tesis es un plug-in en tiempo real.

Hay otro problema relacionado con el tiempo que vale la pena mencionar en este contexto. Si es que una aplicación anfitriona lee muestras de audio de la entrada de la interfaz de audio y sin ningún procesamiento los manda de vuelta a la salida la interfaz de audio, esta tarea toma cierto tiempo para realizarse y habrá un retraso notable entre la señal de entrada y la señal de salida. Este retraso se lo conoce comúnmente como latencia. La latencia depende de varios parámetros tales como la velocidad el CPU, la cantidad de memoria RAM, el tipo de disco duro, tipo de tarjeta de sonido, sistema operativo, el API utilizado para desarrollar el software de audio y el tipo de drivers de la tarjeta de sonido. Esto quiere decir que, el hecho de que un plug-in opere en tiempo real no hace que el sistema operativo se convierta en un procesador de efectos con respuesta en tiempo real comparable con una unidad de efectos dedicada. La latencia dará un retraso notable en la cadena de audio.

5.4 El framework VSTDSK 2.4 para C++

El VSTSDK consiste de un framework C++ orientado a objetos para construir plug-ins VST. Todas las funcionalidades básicas, como por ejemplo la comunicación de buffers de muestras de audio entre la aplicación anfitriona y el plug-in, requeridas por el

plug-in se proveen mediante una clase de base. El código en el VSTDSK especifica la API del plug-in montada sobre la aplicación anfitriona. El desarrollo de plug-ins de audio libera al programador de la tarea de comunicarse directamente con el hardware de interfaz de audio del computador, ya que de esto se encarga la aplicación anfitriona en un nivel inferior, utilizando una de las API's de audio que el driver de la interfaz de audio provee. Una ilustración de esto se puede ver en la figura 5.1.

Figura 5.1. Vista jerárquica de la interfaz de aplicación programable del plug-in [3]

5.5 Código fuente del VSTSDK C++

El código fuente VSTSDK C++ provee un framework bien estructurado para el desarrollo del plug-in. Cualquier persona que sabe de audio digital y de C++ no debería encontrar muchas dificultades para entenderlo. Este hecho y la posibilidad de que, de los mismo archivos del código fuente se puedan crear plug-ins para múltiples plataformas, hace que el tema sea de sumo interés para estudiantes que quieran aprender acerca de DSP en software. El enfoque del trabajo se puede localizar en la parte de audio DSP del código, ya que toda la interacción con el hardware es realizada por la aplicación anfitriona. Para la compañías enfocadas en el desarrollo de plug-ins, la misma base del código de fuente puede ser utilizada para construir plug-ins para MacOS

y Windows; eliminando la necesidad de tener bases de código distintas para las diferentes plataformas. El formato VST es un formato de plug-ins popular, bien establecido en el mercado para programadores independientes. Para plug-ins más complejos, temas como el manejo de información en diferentes plataformas de desarrollo podrían generar problemas y hacer que el código fuente sea específico para cada plataforma, lo cual no se aplica a los plug-ins implementados en esta tesis.

El anfitrión provee una GUI por defecto para el plug-in en el caso de que el programador no haya desarrollado una GUI específica, que permita al usuario ajustar los parámetros de entrada. Gracias a esto, podemos enfocarnos en el código de programación del algoritmo DSP.

5.6 Las clases AudioEffect y AudioEffectX

En 1996 se lanzó la especificación VST 1.0. La versión utilizada por el plug-in en esta tesis es la especificación VST 2.4, la cual es una extensión de la especificación 1.0.

En la especificación VST, la clase base para todos los plug-ins se llama AudioEffect, y se define en un archivo llamado audioeffect.hpp. Partes de este archivo son presentadas a continuación:

```
class AudioEffect
{
public:
  AudioEffect();                          //Constructor
  virtual ~AudioEffect();                 //Destructor

  virtual bool getEffectName();           //Nombre del efecto
  virtual float getParameter();           //Obtiene valor de parametro
  virtual void getParameterDisplay();     //Despliega valor en interfaz grafica (10)
  virtual void getParameterLabel();       //Despliega unidades (dB)
  virtual void getParameterName();        //Despliega (Threshold)
  virtual bool getVendorString();         //Nombre de la empresa desarrolladora
  virtual void setParameter();            //Implementa valor de parametro
  virtual void processReplacing();        //Llamado del host para inserción de efecto
protected:
//    Parametros del compresor
      float fRatio;
      float fThreshold;
      float fAttack;
      float fRelease;
      float fOutputGain;};
```

El código ha sido editado y acortado para aumentar la claridad, y los comentarios son del autor. Solo los métodos y datos más relevantes se muestran. La especificación VST 2.4 extiende la clase base al definir una subclase AufioEffectX en el archivo audioeffectx.h:

```
class AudioEffectX : public AudioEffect
{};
```

Todo el código fue excluido ya que los nuevos métodos y datos miembros no son relevantes para el plug-in en cuestión. AudioEffectX se genera a partir de la clase AudioEffect, por lo tanto es compatible con la especificación VST 1.0. Las clases son implementadas en los archivos audioeffect.cpp y audioeffectx.cpp respectivamente.

Los plug-ins VST tienen dos métodos que contienen el código de procesamientote señal –process() y processReplacing(). El método process() es llamado por el anfitrión cuando el plug-in se utiliza en una configuración de envío auxiliar. Si el plug-in se utiliza como un efecto de inserción en el anfitrión, entonces el método processReplacing() es llamado en su lugar.

5.7 La clase HKCompressor

Para esta explicación solo nos enfocamos en la implementación del plug-in HKCompressor ya que la implementación de los otros tratamientos dinámicos son similares, variando solo el algoritmo de procesamiento el cual será incluido en los anexos. Para construir el plug-in de compresor, una nueva subclase –HKCompressor- fue definida e implementada, la misma que fue generada a partir de AudioEffectX. En esta subclase el método processReplacing() fue redefinido. La jerarquía de clases se muestra en la figura 5.2.

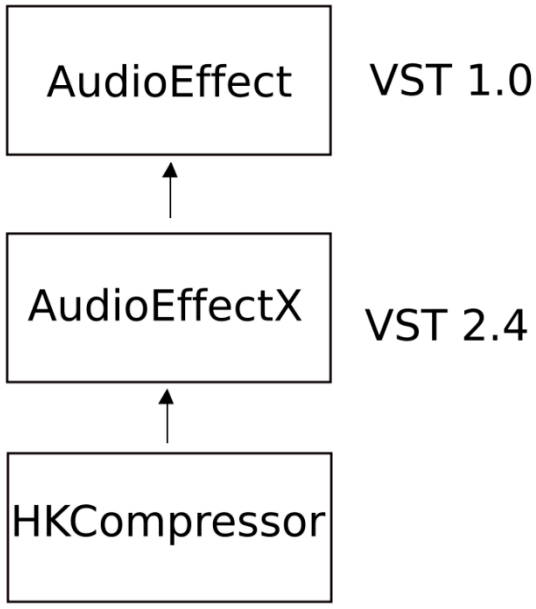


Figura 5.2. Jerarquía de clases para el plug-in VST HKCompressor

La clase HKCompressor se define y es implementada en el método processReplacing() de la siguiente manera

```
class HKCompressor : public AudioEffectX {
public:
  HKCompressor(audioMasterCallback audioMaster);
  ~HKCompressor();

  virtual VstInt32 canDo(char *text);
  virtual bool getEffectName(char* name);

  virtual float getParameter(VstInt32 index);
  virtual void getParameterDisplay(VstInt32 index, char *text);
  virtual void getParameterLabel(VstInt32 index, char *label);
  virtual void getParameterName(VstInt32 index, char *text);
  virtual VstPlugCategory getPlugCategory();

  virtual bool getProductString(char* text);
  virtual void getProgramName(char *name);
  virtual bool getVendorString(char* text);
  virtual VstInt32 getVendorVersion();

  virtual void processReplacing(float **inputs, float **outputs, VstInt32 sampleFrames);

  virtual void setParameter(VstInt32 index, float value);
  virtual void setProgramName(char *name);

protected:
      float fRatio;
      float fThreshold;
      float fAttack;
      float fRelease;
      float fOutputGain;
};
```

El código mostrado no está editado. Solo se dejaron fuera alguna definiciones (#define) y enumeraciones (enum) que se encuentran antes de la declaración de la clase. El código que se encarga realmente del procesamiento, es implementado en el método processReplacing() de la siguiente manera:

```
void HKCompressor::processReplacing(float **inputs, float **outputs, VstInt32
sampleFrames) {

	float* in = inputs[0];
	float* out = outputs[0];

	//=============================================//
	//    Inicializacion de variable del algoritmo  //
	//=============================================//

	float log_10;
	float logrms=0.0;
	float sg=0.0;
	float dg=1.0;
	int R,thdB;
	float th,AT,RT,x,y;

	//================================//
	//    Parametros del compresor    //
	//================================//

	float fs = 44100.0;             //Frecuencia de muestreo
	float T = 1.0 / fs;            //Periodo de muestreo
	float TRT = 0.1e-3;           //Triggering time
	float TR = 1 - exp(-2.2 * T / TRT); // Triggering time
	R = (fRatio * 29) + 1;
	thdB = (fThreshold) * 50;


	//=========================//
	//    Ajuste threshold     //
	//=========================//


	 th = pow(10,-thdB/20) * log10(2); // Ajuste logaritmico de threshold


	//=========== Calculo de la ganancia estatica y dinamica ===============//

//    Ganancia estatica: Se deriva de la grafica de la funcion de
//    transferencia de un compresor.
//    La formula de esta grafica es: Y = TH + (X - TH) / R.
//    En el dominio logaritmico la funcion de transferencia equivale a la
//    resta de Y - X.  De donde derivamos que la funci?n de control del
//    compresor en dominio logar?tmico es SG = (X - TH) / R + TH - X.
//
//    Ganancia dinamica:  Se deriva de la ecuacion a diferencias del grafico
//    del libro de Zolzer.
//    La ecuacion es: Y[n] = AT * SG[n] + Y[n-1] * (1-AT) para attack time
//    y, Y[n] = RT * SG[n] + Y[n-1] * (1-RT) para release time.

	while (--sampleFrames >= 0)
	{
	  AT = exp(-2.0 * T / ((fAttack * 0.199)+0.001)); // Attack time
	  RT = exp(-2.0 * T / ((fRelease * 4.999)+0.001)); // Release time
	  x = *in++;
	  y = x;
	  y = fabs(y);                //Valor absoluto de y
	  log_10 = log10(y + 1);     //Offset para evitar log(0)
	  logrms = (TR * (2.0 * log_10) + logrms * (1 - TR)) / 2.0; //Calculo valor RMS
	  if (logrms >= th) {
```

```
                // Ganancia estatica
                sg = pow(10,(((logrms-th)/R+th-logrms)/log10(2)));
                // Ganancia dinamica en attack time
                dg = AT * sg + (1 - AT) * dg;
        } else {

                // Ganancia dinamica en release time
                dg = RT * sg + (1 - RT) * dg;
                sg = 1;
        }
        *out++ = x * dg;
      }

}
```

Solo el método processReplacing() ha sido mostrado aquí, ya que la manera normal de implementar un compresor es insertándolo en el camino de la señal. El listado de código C++ de esta sección ha sido reducido a su mínimo para que las ideas generales sean más claras sin ser entorpecidas por muchos detalles. Es importante recalcar que por defecto el valor de los parámetros varía entre 0 y 1, es por esta razón que hay que escalar este rango de valores al rango apropiado para cada uno de los parámetros.

Finalmente, en lo que respecta la interfaz gráfica de usuario, toda la información desplegada en la misma es implementada en los métodos antes mostrados en la definición de la clase HKCompresor. Estos métodos sirven para la obtención y despliegue de toda la información, es decir para la comunicación entre plug-in y usuario:

```
AudioEffect* createEffectInstance(audioMasterCallback audioMaster) {
      return new HKCompressor (audioMaster);
}
//Constructor
HKCompressor::HKCompressor(audioMasterCallback audioMaster)
: AudioEffectX(audioMaster, kNumPrograms, kNumParameters) {
//Valores por default del plug-in
      fRatio = 0;
      fThreshold = 0;
      fAttack = 0.05;
      fRelease = 0.01;
      fOutputGain =0.25;
      setProgram (1);
    setNumInputs(1);
    setNumOutputs(1);
    setUniqueID('HKCP');
}

HKCompressor::~HKCompressor() {    //Destructor
}

bool HKCompressor::getEffectName(char* name) {
  strncpy(name, "HK Compressor", kVstMaxProductStrLen);
  return true;
}
```

```
float HKCompressor::getParameter(VstInt32 index) {
      //Obtiene el valor de parametro
      float v = 0;
      switch (index)
      {
        case kRatio :       v = fRatio;             break;
        case kThreshold :   v = fThreshold;         break;
        case kAttack :      v = fAttack;            break;
        case kRelease :     v = fRelease;           break;
        case kOutputGain :  v = fOutputGain;        break;
      }
      return v;
}

void HKCompressor::getParameterDisplay(VstInt32 index, char *text) {
      //Despliega valor de parametro escalado a rango apropiado
      switch (index)
      {
        case kRatio :int2string ((fRatio * 29) + 1, text, kVstMaxParamStrLen);
      break;
        case kThreshold :int2string ((fThreshold) * 50, text, kVstMaxParamStrLen);
      break;
        case kAttack :((fAttack * 199) + 1, text, kVstMaxParamStrLen);
      break;
        case kRelease :int2string ((fRelease * 4999) + 1, text, kVstMaxParamStrLen);
      break;
        case kOutputGain : dB2string (fOutputGain*4, text, kVstMaxParamStrLen);
      break;
      }
}

void HKCompressor::getParameterLabel(VstInt32 index, char *label) {
      //Obtiene y despliega unidad de parametros
      switch (index)
      {
        case kRatio :      strcpy (label, ":1");       break;
        case kThreshold :  strcpy (label, "dB");       break;
        case kAttack :     strcpy (label, "ms");       break;
        case kRelease :    strcpy (label, "ms");       break;
        case kOutputGain : strcpy (label, "dB");       break;
      }
}

void HKCompressor::getParameterName(VstInt32 index, char *label) {
      //Obtiene y despliega nombres de parametros
      switch (index)
      {
        case kRatio :      strcpy (label, "Ratio");           break;
        case kThreshold :  strcpy (label, "Threshold");       break;
        case kAttack :     strcpy (label, "Attack Time");     break;
        case kRelease :    strcpy (label, "Release Time");    break;
        case kOutputGain:  strcpy (label, "Output Gain");     break;
      }
}

bool HKCompressor::getProductString(char* text) {
  strcpy(text, "HK Compressor");
  return true;
}

bool HKCompressor::getVendorString(char* text) {
  strcpy(text, "Hell's Kitchen");
  return true;
}

VstInt32 HKCompressor::getVendorVersion() {
      return 1000;
}
```

```
void HKCompressor::setParameter(VstInt32 index, float value) {
      //Implementa valor de parametros
      switch (index)
      {
        case kRatio :       fRatio = value;          break;
        case kThreshold :   fThreshold = value;      break;
        case kAttack :      fAttack = value;         break;
        case kRelease :     fRelease = value;        break;
        case kOutputGain :  fOutputGain = value;     break;
      }
}

}
```

Capítulo 6 Resultados y pruebas en aplicaciones anfitrionas

6.1 Resultados

El plug-in VST fue probado en diferentes aplicaciones anfitrionas VST en MacOS. Las pruebas en Mac OS se realizaron en una MacBook Pro con las siguientes especificaciones:

- Procesador Intel Core Duo 2 2.8GHz
- 4 GB 1067 MHz DDR3
- Sistema operativo Mac OS X 10.6.3
- Tarjeta de audio Digidesign Mbox Pro

El plug-in apareció sin problemas en todas las aplicaciones anfitrionas. Se comprobó el funcionamiento del plug-in al reproducir archivos de audio del disco duro y aplicando el plug-in en todas las aplicaciones probadas. De esta manera se pudieron hacer constataciones de cómo el procesamiento del plug-in afectaba la señal de audio.

Para medir el desempeño del plug-in de manera objetiva se utilizó Matlab para analizar las señales procesadas por los plug-ins funcionando dentro de las diferentes aplicaciones anfitrionas. Al grabar diferentes señales para un set específico de parámetros se pudo comparar los resultados de los prototipos hechos en Matlab con los algoritmos implementados en tiempo real.

Finalmente la GUI por defecto resultante del plug-in fue comparada entre las diferentes aplicaciones anfitrionas: Cubase, AudioMulch, VSTiHost y Logic.

6.2 Constatación auditiva

Durante el desarrollo del prototipo en Matlab, se escuchó el desempeño de los tratamientos dinámicos. También se escuchó al plug-in VST compilado en todas las aplicaciones anfitrionas probadas. Utilizando música y grabaciones a distintos instrumento como señal de entrada, se confirmó auditivamente que los plug-ins afectaban la señal procesada y cumplían globalmente con sus funciones.

6.3 Evaluación de resultados en Matlab

En esta sección se presentan los resultados de las señales procesadas por los plug-ins. Estas señales fueron grabadas en tiempo real por la aplicación anfitriona y guardadas como archivos wave parar poder ser gráficadas en Matlab. La señal de entrada está representada en color azul y la señal de salida en color rojo.

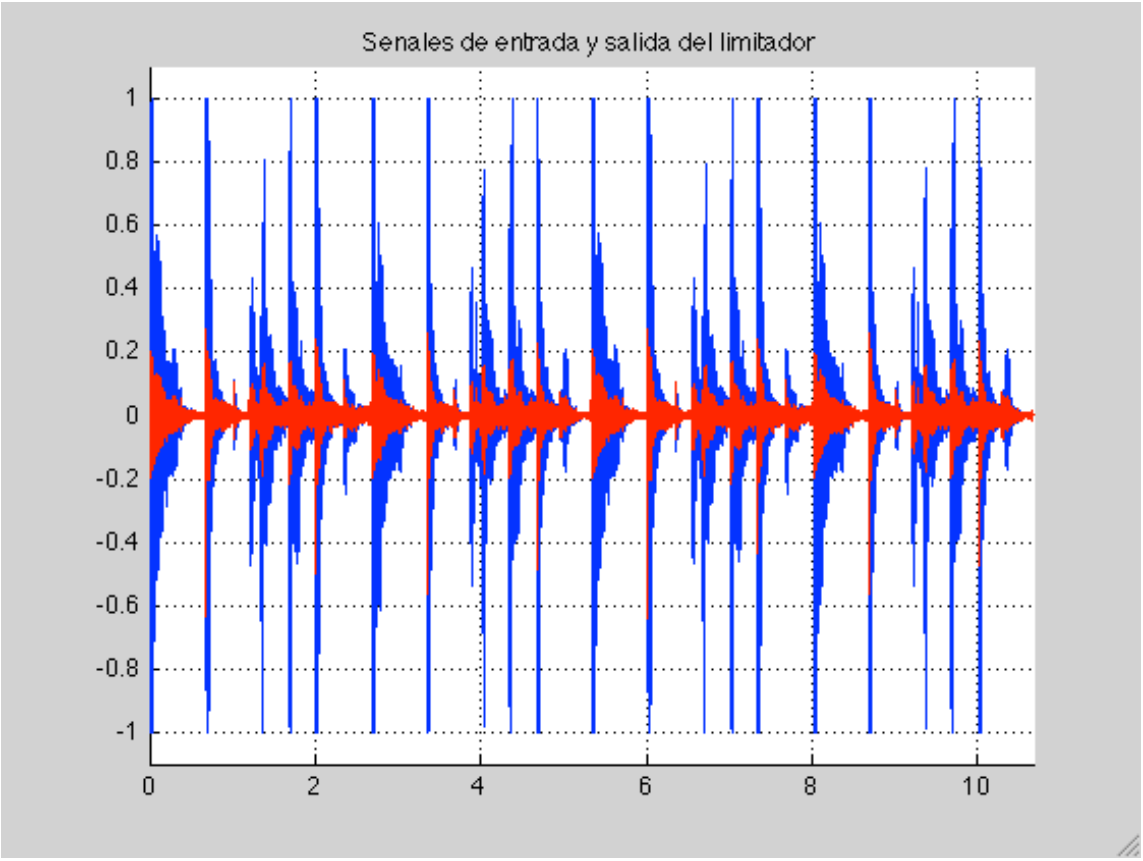


Figura 6.1 Evaluación de plug-in limitador (HKLimiter)

Vemos, en la figura 6.1, como el paso de la señal por el plug-in tiene como resultado una reducción considerable de la dinámica de la señal.

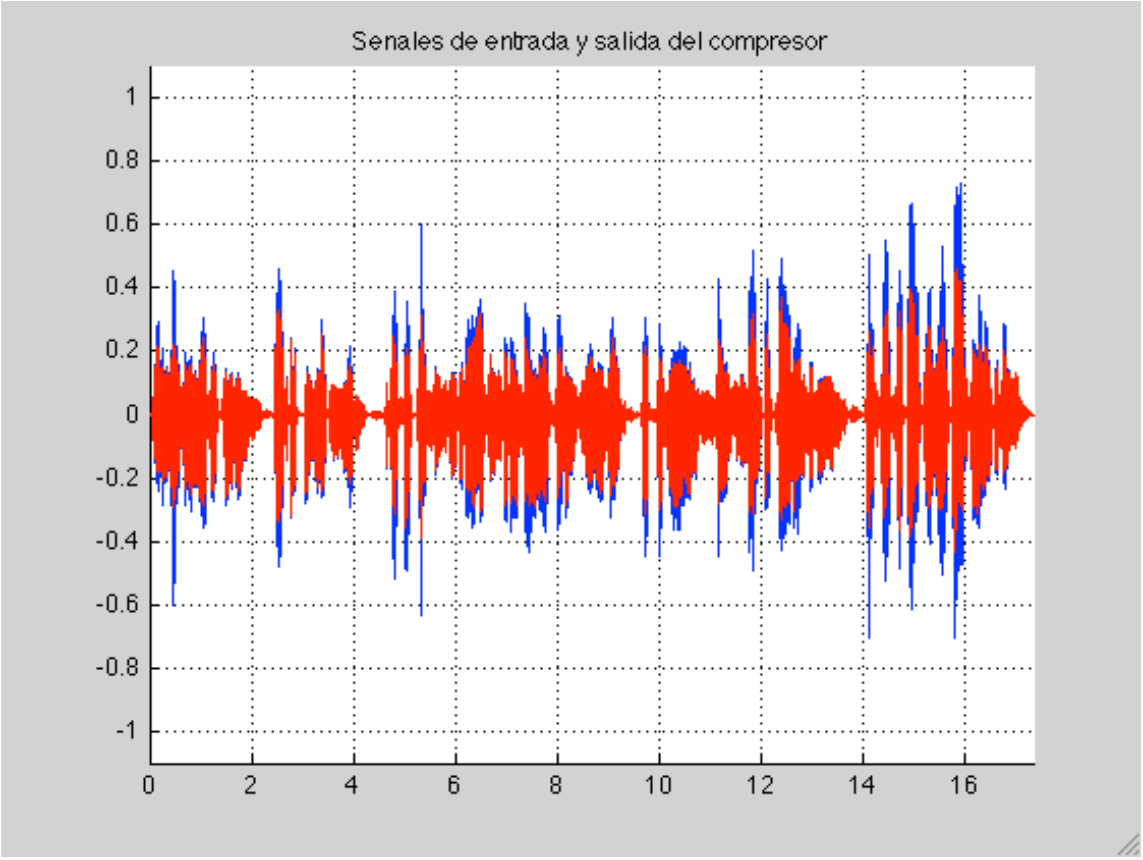


Figura 6.2 Evaluación de plug-in compresor (HK Compresor)

En la figura 6.2 vemos los resultados de una compresión de una señal de audio de voz. El plug-in actúa sobre los nivele que sobrepasan el umbral y comprime, de acuerdo a los parámetros de tiempo, la señal de salida.

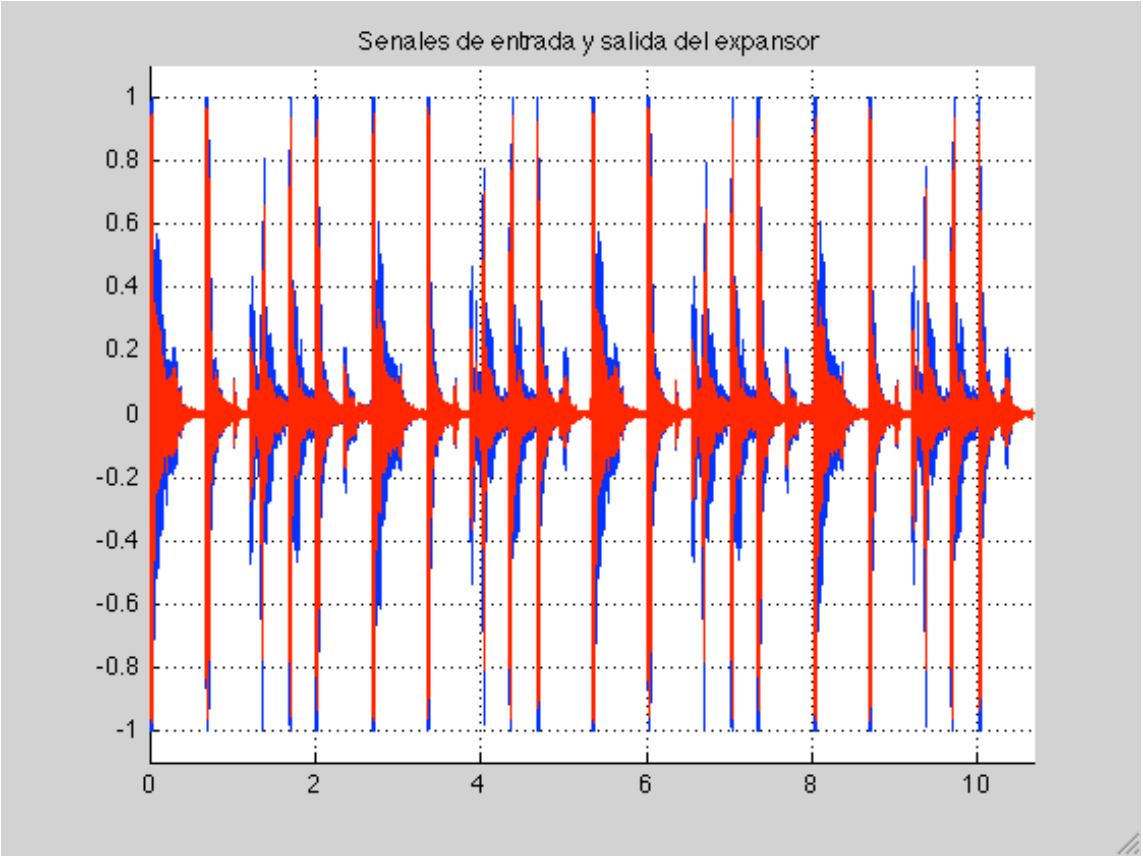


Figura 6.3 Evaluación de plug-in expansor (HKExpander)

En las figuras 6.3 y 6.4 se pueden ver los resultados del expansor y la compuerta de ruido respectivamente. Vemos que las acciones del expansor y la compuerta de ruido son similares. Como lo mencionamos antes la compuerta de ruido es un caso especial de un expansor en donde la relación entre la salida y la entrada es infinita.

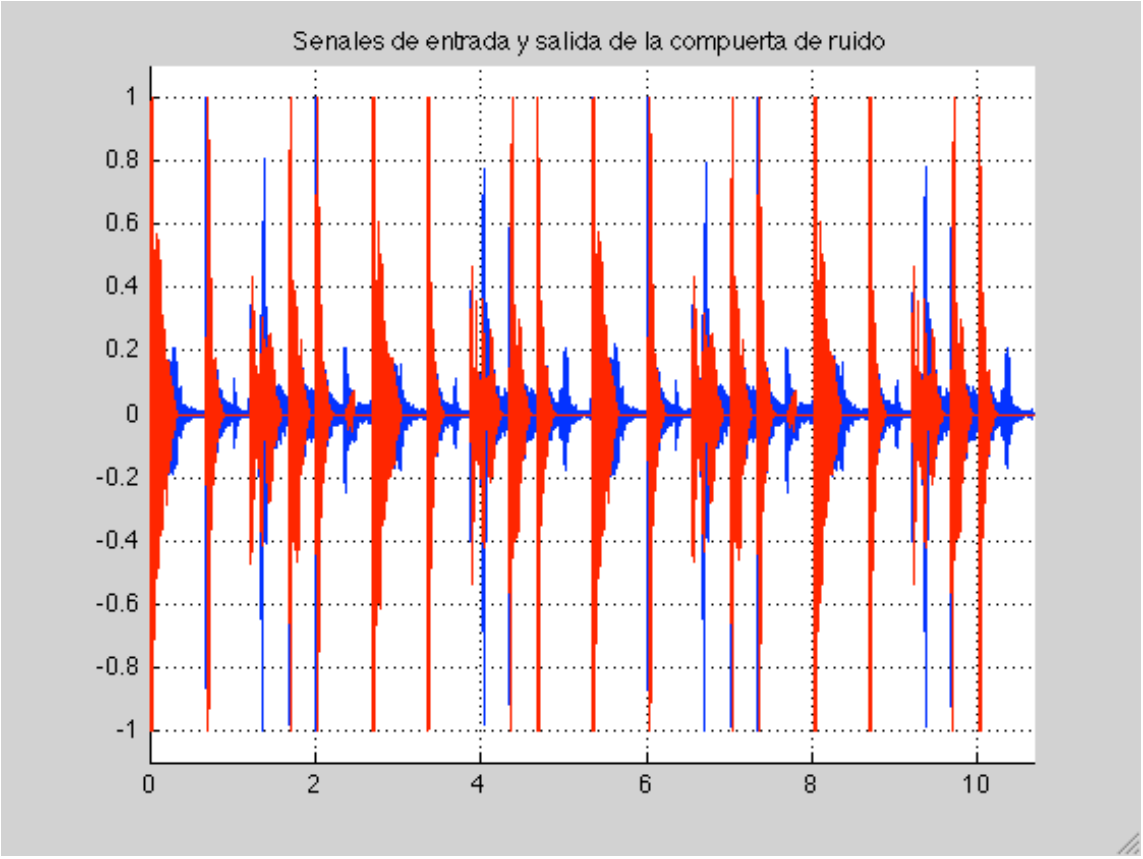


Figura 6.4 Evaluación de plug-in compuerta de ruido (HKNoiseGate)

6.4 Aplicaciones anfitrionas en Mac OS

Ya que la implementación presentada en esta tesis no trató con GUI del plug-in, la representación gráfica varió entre las diferentes aplicaciones anfitrionas como lo veremos a continuación.

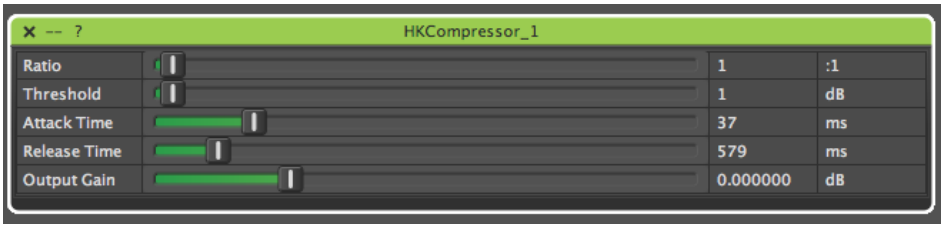


Figura 6.5. GUI de Audiomulch

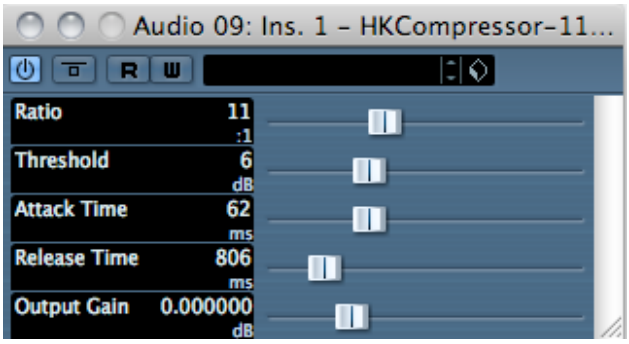


Figura 6.6. GUI de Cubase LE 4

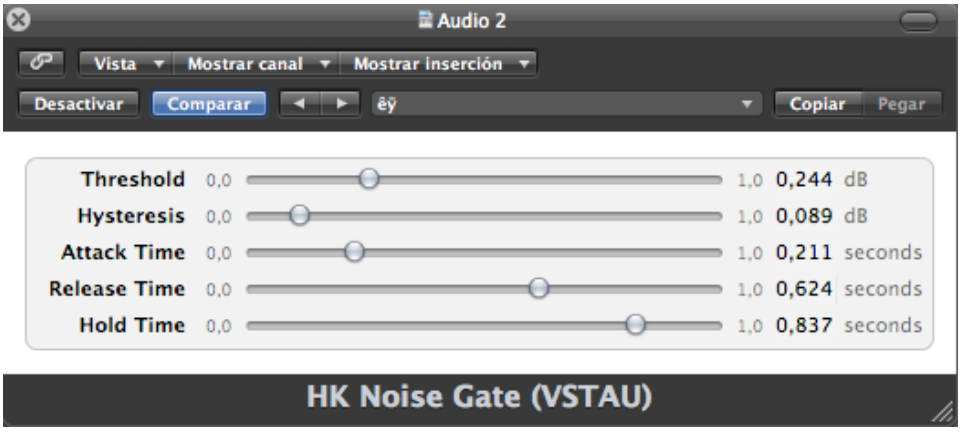


Figura 6.7. GUI de Logic Express 8

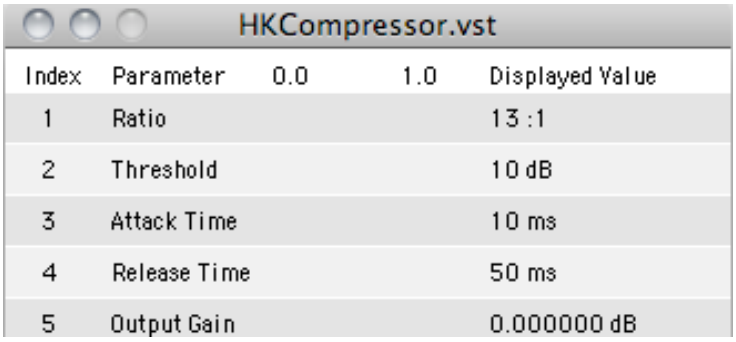


Figura 6.8. GUI VST Host Demo

Como podemos constatar, las interfaces gráficas varían en aspecto estético más no en su manera global de desplegar los parámetros. Sin embargo vale la pena destacar que en la aplicación Logic, el plug-in no puede mostrar los valores apropiados para sus

parámetros. Es decir que el rango de valores que se despliega varía entre 0 y 1. Internamente el plug-in funciona con el rango de valores que el programador le asignó pero para su despliegue no funciona de la misma manera. Esto se debe a que en realidad el plug-in en Logic no es un plug-in VST sino un plug-in VST envuelto en un plug-in AU, gracias al software VSTAU Manager. Podemos deducir que el momento de envolver un formato dentro del otro información valiosa como el despliegue de parámetros, no es traducida de buena manera.

Adicionalmente podemos decir que en el caso de la aplicación VST Host Demos, siendo una aplicación muy básica, el anfitrión ni siquiera nos provee un cursor para variar los parámetros. Estos se pueden modificar con el mouse pero de manera invisible.

Capítulo 7 Conclusiones

7.1 Optimización

El código C++ escrito para este plug-in no fue optimizado de ninguna manera para velocidad o tamaño. Simplemente fue una implementación directa de un filtro y las fórmulas necesarias para calcular este tratamiento. De esta manera el código es más una manera de ilustración de los principios básicos de programación de plug-ins, que un plug-in altamente optimizado y de alto rendimiento.

Trabajos futuros en este tema podrían apuntar a la reducción de la carga del CPU generada por este plug-in. Esta es un área importante en general, ya que todos los plug-ins en una aplicación anfitriona, comparten la misma cantidad finita de capacidad del CPU disponible; y una menor carga del CPU por plug-in significa que más plug-ins pueden ser utilizados a la vez.

7.2 Interplataformas de la GUI

Como muestran los gráficos de la sección 6.4, la apariencia del GUI por defecto del plug-in difiere considerablemente entre las aplicaciones anfitrionas. Adicionando un código para el manejo customizado de la GUI se puede lograr mantener la misma apariencia sin importar la aplicación anfitriona utilizada.

La programación de la GUI en general tiende a ser compleja y se inclina a la especificidad de plataformas. Dentro del VSTSDK se encuentran bibliotecas VST GUI que pueden ser usadas para crear y manejar una GUI con faders, perillas, diales, etc. Esto nos ofrece una manera de implementar GUIs sin tener que lidiar con detalles específicos de cada plataforma.

7.3 Conclusiones y recomendaciones

Esta tesis se enfocó en el desarrollo de un plug-in de audio VST. Al comenzar con un prototipo hecho en un medio de programación gráfico se obtuvo la experiencia necesaria para la futura implementación de los algoritmos. Los algoritmos funcionaron muy bien en las simulaciones realizadas en Matlab. Se realizaron pruebas con distintos tipos de señales para poder ver con precisión los efectos del tratamiento dinámico en una señal de audio musical. Lo aprendido fue luego aplicado en la implementación del plug-in en C++. En adición a este método de desarrollo de dos etapas, la tesis también demostró un método de verificación del procesamiento de audio del plug-in final, al analizar los resultados en Matlab. Debido al trabajo algorítmico de preparación conducido en el prototipo, la implementación inicialmente se realizó sin mayores complicaciones. Sin embargo, al momento de poner a prueba los plug-ins se pusieron en evidencia ciertas limitaciones de los algoritmos implementados sobre todo para la aplicación de constantes de tiempo que controlan los plug-ins. Esto refleja que el hecho de que los algoritmos funcionen de manera muy eficiente en prototipos y simulaciones no quiere decir que funcionen necesariamente de la misma manera una vez implementados para tratamiento en tiempo real. Esto es independiente de la aplicación anfitriona que se utilice ya que con todas las aplicaciones probadas se obtuvieron los mismos resultados. En realidad se debe principalmente a las limitaciones con el lenguaje de programación y en el conocimiento de métodos de optimización de algoritmos DSP, lo que, junto con la creación de un GUI customizado, están fuera de la alcance de esta tesis, pero podría ser un camino interesante para futuros proyectos.

Una primera alternativa que se intentó llevar a cabo fue la implementación de los algoritmos en una tarjeta DSP, lo que vale la pena decir que es muy recomendable ya que se cuenta con un procesador dedicado exclusivamente a la ejecución del algoritmo y

no otras tareas que lleva a cabo una computadora. Lamentablemente, si bien es cierto la Universidad se cuenta con una docena de tarjetas DSP Motorola, al momento no se disponen de las licencias y por ende no se les puede dar uso alguno. Sería interesante adquirir estas licencias para que en un futuro otros alumnos interesados en el tratamiento de señales de audio puedan experimentar con ellas.

El desarrollo e implementación de algoritmos DSP requiere de una investigación profunda y puede ser objeto de toda una carrera profesional, ya que se necesita tener conocimientos extensos en dos dominios de la ingeniería como son la electrónica y la programación de sistemas.

Bibliografía:

[1] Zölzer, Udo. *DAFX*. John Wiley & Sons, 2002.

[2] Zölzer, Udo. *Digital Audio Signal Processing*. John Wiley & Sons. 2nd ed, 2008.

[3] Steinberg VST 2.4 Software Development Kit (SDK)
URL: http://www.steinberg.net/en/company/developer.html

[4] Arfib, Daniel. *Different ways to write digital audio effects programs.* Barcelona, España, 2001.
URL: member.sn7.de/niklas/mt/Arfib-DAFX_Programming.pdf

[5] Floru, Fred. Attack and Release time constants in RMS-based Feedback Compressors. AES 104th Convention. Amsterdam, 1998.

[6] Floru, Fred. Attack and Release time constants in RMS-based Compressors and Limiters. AES 99th Convention. New York, 1995.

[7] McNally, G.W. Dynamic Range Control of Digital Audio Signals. J. Audio Eng. Soc., Vol. 32, No. 5,1984.

[8] Perry, Tim. *Dynamics Processors: Limiter, Compressor/Expander, Noise Gate.* Audio Signal Processing Assignment. University of Victoria, Junio 2009.

[9] Roads, Curtis. *The Computer Music Tutorial.* The MIT Press, Cambridge, Massachusetts.

[10] Ballou, Glen, *Handbook for sound engineers,*Focal Press. Fourth edition, 2008.

[11] Oppenheim, A., Schafer, R. *Discrete-time signal processing.* Prentice Hall, 2nd ed. 1999.

[12] Mettens, Richard. *Efectos,* Instituto SAE Paris carrera de Audio Engineering, 2009-2010

[13] THAT Corporation
URL: http://www.thatcorp.com

Anexos 1:

Implementación analógica

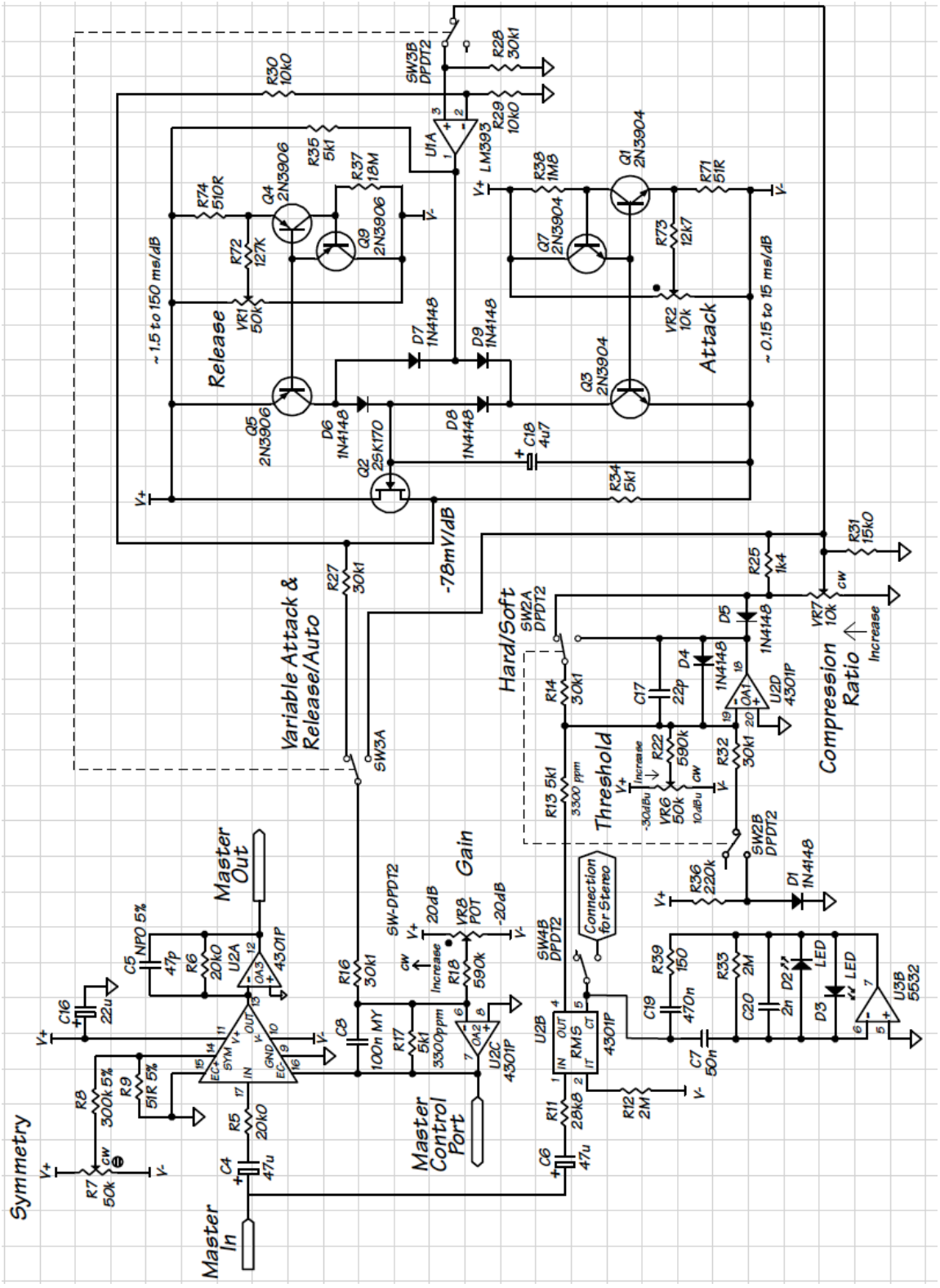
Inicialmente se intentó construir un prototipo analógico. Se realizó una investigación previa para determinar los componentes y etapas necesarias para la construcción de un compresor analógico. Dado la complejidad de la construcción de este prototipo se decidió adquirir un chip de THAT Corporation (THAT 4301[6]) [13] que no está de venta en el país, el cual supuestamente facilitaría el proceso de la construcción de este prototipo. Sin embargo, el tratamiento de señales de audio debe ser extremadamente delicado y preciso ya que cualquier ruido inducido por el proceso es audible. La exigencia a nivel de precisión de estos diseños hace que los componentes utilizados en estos circuitos tengan una tolerancia muy baja, desde resistencias hasta capacitores. En su gran mayoría los componentes requeridos en el diseño que se intentó implementar no están disponibles en el mercado local. Además, una limitación adicional, a nivel de la construcción del circuito, fue la dificultad para implementar una placa con chips pequeños los cuales deben ser soldados sobre la superficie de la placa. Realizar una tarea semejante con los métodos tradicionales (impresión de layouts, utilización de papel termotransferible) resulta difícil por la precisión que debe tener el diseño en una placa de doble lado. A pesar de esto se construyó el prototipo [7] sin ningún resultado audible. El tiempo dedicado a esta construcción fue extenso y debido a su complejidad se decidió dedicar esta tesis hacer una implementación en el dominio digital.

---

[6] http://www.thatcorp.com. Datasheet en anexos 1

[7] Layout y fotos de placa construida en anexos 1

Circuito analógico implementado[8]:



[8] http://www.thatcorp.com/4301_Analog_Engine_Dynamics_Processor.shtml

Datasheet Chip THAT 4301[9]:

THAT Corporation

THAT Analog Engine®
IC Dynamics Processor

THAT 4301, 4301A

FEATURES

- High-Performance Voltage Controlled Amplifier
- High-Performance RMS-Level Detector
- Three General-Purpose Opamps
- Wide Dynamic Range: >115 dB
- Low THD: <0.03%
- Low Cost
- DIP & Surface-Mount Packages

APPLICATIONS

- Compressors
- Limiters
- Gates
- Expanders
- De-Essers
- Duckers
- Noise Reduction Systems
- Wide-Range Level Meters

Description

THAT 4301 Dynamics Processor, dubbed "THAT Analog Engine," combines in a single IC all the active circuitry needed to construct a wide range of dynamics processors. The 4301 includes a high-performance, exponentially-controlled VCA, a log-responding RMS-level sensor and three general- purpose opamps.

The VCA provides two opposing-polarity, voltage-sensitive control ports. Dynamic range exceeds 115 dB, and THD is typically 0.003% at 0 dB gain. In the 4301A, the VCA is selected for low THD at extremely high levels. The RMS detector provides accurate rms-to-dc conversion over an 80 dB dynamic range for signals with crest factors up to 10. One opamp is dedicated as a current-to-voltage converter for the VCA, while the other two may be used for the signal path or control voltage processing.

The combination of exponential VCA gain control and logarithmic detector response — "decibel-linear" response — simplifies the mathematics of designing the control paths of dynamics processors. This makes it easy to design audio compressors, limiters, gates, expanders, de-essers, duckers, noise reduction systems and the like. The high level of integration ensures excellent temperature tracking between the VCA and the detector, while minimizing the external parts count.

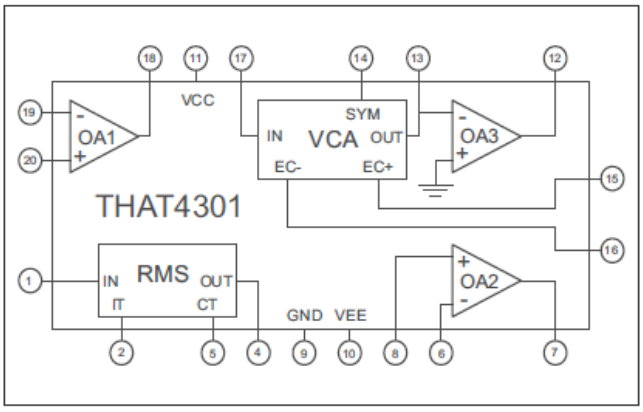


Figure 1. Block Diagram (pin numbers are for DIP only)

Model	20 pin DIP Pkg	30 pin DMP (SO) Pkg
4301 RoHS Compliant	4301P20-I	4301M30-I
4301A RoHS Compliant	4301AP20-I	---
4301	---	4301M30
4301A	4301AP20	---

Table 1. Ordering Information

THAT Corporation; 45 Sumner Street; Milford, Massachusetts 01757-1656; USA
Tel: +1 (508) 478-9200; Fax: +1 (508) 478-0990; Web: www.thatcorp.com
Document 600069 Rev. 03

[9] http://www.thatcorp.com/4301_Analog_Engine_Dynamics_Processor.shtml

SPECIFICATIONS[1,2]

Absolute Maximum Ratings (T_A = 25°C)

Positive Supply Voltage (V_{CC})	+18 V	Power Dissipation (P_D) (T_A = 75°C)	700 mW
Negative Supply Voltage (V_{EE})	-18 V	Operating Temperature Range (T_{OP})	0 to +70°C
Supply Current (I_{CC})	20 mA	Storage Temperature Range (T_{ST})	-40 to +125°C

Overall Electrical Characteristics

Parameter	Symbol	Conditions	Min	Typ	Max	Units
Positive Supply Voltage	V_{CC}		+7	—	+15	V
Negative Supply Voltage	V_{EE}		-7	—	-15	V
Positive Supply Current	I_{CC}		—	12	18	mA
Negative Supply Current	I_{EE}		—	-12	-18	mA
Thermal Resistance	θ_{J-C}	SO-Package	—	140	—	°C/W

VCA Electrical Characteristics[3]

			4301			4301A			
Parameter	Symbol	Conditions	Min	Typ	Max	Min	Typ	Max	Units
Input Bias Current	$I_{B(VCA)}$	No Signal	—	30	400	—	30	400	pA
Input Offset Voltage	$V_{OFF(VCA\ In)}$	No Signal	—	±4	±15	—	±4	±15	mV
Input Signal Current	$I_{IN(VCA)}$ or $I_{OUT(VCA)}$		—	175	750	—	175	750	µArms
Gain at 0V Control	G_0	$E_{C+} = E_{C-} = 0.000V$	-0.4	0.0	+0.4	-0.4	0.0	+0.4	dB
Gain-Control Constant		T_A = 25°C ($T_{CHIP} \cong$ 55°C) -60 dB < gain < +40dB							
	E_{C+}/Gain (dB)	E_{C+} & SYM	6.4	6.5	6.6	6.4	6.5	6.6	mV/dB
	E_{C-}/Gain (dB)	E_{C-}	-6.4	-6.5	-6.6	-6.4	-6.5	-6.6	mV/dB
Gain-Control TempCo	$\Delta E_C / \Delta T_{CHIP}$	Ref T_{CHIP}= 27°C	—	+0.33	—	—	+0.33	—	%/°C
Gain-Control Linearity		-60 to +40 dB gain	—	0.5	2	—	0.5	2	%
Off Isolation		E_{C+}=SYM=-375mV, E_{C-}=+375mV	110	115	—	110	115	—	dB
Output Offset Voltage Change	$\Delta V_{OFF(OUT)}$	R_{out} = 20kΩ							
		0 dB gain	—	1	3	—	1	3	mV
		+15 dB gain	—	2	10	—	2	10	mV
		+30 dB gain	—	5	25	—	5	25	mV
Gain Cell Idling Current	I_{IDLE}		—	20	—	—	20	—	µA
Output Noise	$e_{n(OUT)}$	20 Hz-20 kHz							
		R_{out} = 20kΩ							
		0 dB gain	—	-96	-94	—	-96	-94	dBV
		+15 dB gain	—	-85	-83	—	-85	-83	dBV
Total Harmonic Distortion	THD	V_{IN}= 0 dBV, 1 kHz							
		0 dB gain	—	0.003	0.007	—	0.003	0.007	%

1. All specifications subject to change without notice.
2. Unless otherwise noted, T_A=25°C, V_{CC} = +15V, V_{EE}= -15V; VCA_{SYM} adjusted for min THD @ 1 V, 1 kHz, 0 dB gain.
3. Test circuit is the VCA section only from Figure 2.

THAT Corporation; 45 Sumner Street; Milford, Massachusetts 01757-1656; USA
Tel: +1 (508) 478-9200; Fax: +1 (508) 478-0990; Web: www.thatcorp.com

SPECIFICATIONS[1,2] (Cont'd.)

VCA Electrical Characteristics[3] (Cont'd.)

Parameter	Symbol	Conditions	4301 Min	4301 Typ	4301 Max	4301A Min	4301A Typ	4301A Max	Units
Total Harmonic Distortion (cont'd.)	THD	V_{IN} = +10 dBV, 1 kHz							
		0 dB gain	—	0.03	0.07	—	0.03	0.07	%
		–15 dB gain	—	0.035	0.09	—	0.035	0.09	%
		V_{OUT}= +10 dBV, 1 kHz							
		+15 dB gain	—	0.035	0.09	—	0.035	0.09	%
		V_{IN} = +19.5 dBV, 1 kHz							
		0 dB gain	—	—	—	—	0.05	0.09	%
Symmetry Control Voltage	V_{SYM}	minimum THD	-2.5	0	+2.5	-2.5	0	+2.5	mV

RMS Detector Electrical Characteristics[4]

Parameter	Symbol	Conditions	Min	Typ	Max	Units
Input Bias Current	$I_{B\ (RMS)}$	No Signal	—	30	400	pA
Input Offset Voltage	$V_{OFF(RMS\ In)}$	No Signal	—	±4	±15	mV
Input Signal Current	$I_{IN(RMS)}$		—	175	750	µA
Input Current for 0 V Output	I_{in0}	I_T= 7.5µA	6	8.5	12	µA
Output Scale Factor	E_O / 20log(I_{in}/I_{in0})	31.6nA< I_{IN}< 1mA				
		T_A= 25°C (T_{CHIP} ≈ 55°C)	6.4	6.5	6.6	mV/dB
Scale Factor Match (RMS to VCA)		-20 dB < VCA Gain < +20 dB				
		1µA<$I_{in\ (DET)}$<100µA	.985	1	1.015	
Output Linearity		f_{IN}= 1kHz				
		1µA < I_{in}< 100µA	—	0.1	—	dB
		100nA < I_{in}< 316µA	—	0.5	—	dB
		31.6nA < I_{in}< 1mA	—	1.5	—	dB
Rectifier Balance		f_{IN} = 100 Hz, τ = .001 s				
		1µA< I_{in} < 100µA	–20	—	20	%
Crest Factor		1ms pulse repetition rate				
		0.2 dB error	—	3.5	—	
		0.5 dB error	—	5	—	
		1.0 dB error	—	10	—	
Maximum Frequency for 2 dB Additional Error		I_{in} ≥ 10µA	—	100	—	kHz
		I_{in} ≥ 3µA	—	45	—	kHz
		I_{in} ≥ 300nA	—	7	—	kHz
Timing Current Set Range	I_T		1.5	7.5	15	µA
Voltage at I_T Pin		I_T = 7.5µA	-10	+20	+50	mV
Timing Current Accuracy	I_{CT}/I_T	I_T = 7.5µA	0.90	1.1	1.30	
Filtering Time Constant	τ	T_{CHIP} = 55°C		$(0.026)^{C_T}/_{I_T}$		s
Output Temp. Coefficient	$\Delta E_o / \Delta T_{CHIP}$	Re: T_{CHIP} = 27°C	—	0.33	—	%/°C
Output Current	I_{OUT}	–300mV < V_{OUT}< +300mV	±90	±100	—	µA

4. Except as noted, test circuit is the RMS-Detector section only from Figure 2.

THAT Corporation; 45 Sumner Street; Milford, Massachusetts 01757-1656; USA
Tel: +1 (508) 478-9200; Fax: +1 (508) 478-0990; Web: www.thatcorp.com

Specifications[1,2] (Cont'd)

Opamp Electrical Characteristics[5]

			OA1			OA2			OA3			
Parameter	Symbol	Conditions	Min	Typ	Max	Min	Typ	Max	Min	Typ	Max	Units
Input Offset Voltage	V_{OS}		—	±0.5	±6	—	±0.5	±6	—	±0.5	±6	mV
Input Bias Current	I_B		—	150	500	—	150	500	—	150	500	nA
Input Offset Current	I_{OS}		—	15	50	—	15	50		N/A		nA
Input Voltage Range	I_{VR}		—	±13.5	—	—	±13.5	—		N/A		V
Common Mode Rej. Ratio	CMRR	R_S<10k	—	100	—	—	100	—		N/A		
Power Supply Rej. Ratio	PSRR	VS=±7V to ±15V	—	100	—	—	100	—	—	100	—	
Gain Bandwidth Product	GBW	(@50kHz)	—	5	—	—	5	—	—	5	—	MHz
Open Loop Gain	A_{VO}	R_L=10k	—	115	—	—	110	—	—	125	—	
		R_L=2k		N/A			N/A		—	120	—	
Output Voltage Swing	V_O@R_L=5kΩ		—	±13	—	—	±13	—	—	±14	—	V
	V_O@R_L=2kΩ			N/A			N/A	—	—	±13	—	V
Short Circuit Output Current			—	4	—	—	4	—	—	12	—	mA
Slew Rate	SR		—	2	—	—	2	—	—	2	—	V/µs
Total Harmonic Distortion	THD	1kHz, A_V=1, R_L=10kΩ	—	0.0007	0.003	—	0.0007	0.003	—	0.0007	0.003	%
		1kHz, A_V=–1, R_L= 2kΩ		N/A			N/A		—	0.0007	0.003	%
Input Noise Voltage Density	e_n	f_O=1kHz	—	6.5	10	—	7.5	12	—	7.5	12	$nV/\sqrt{Hz}$
Input Noise Current Density	i_n	f_O=1kHz	—	0.3	—	—	0.3	—	—	0.3	—	$pA/\sqrt{Hz}$

5. Test circuit for opamps is a unity-gain follower configuration, with load resistor R_L as specified.

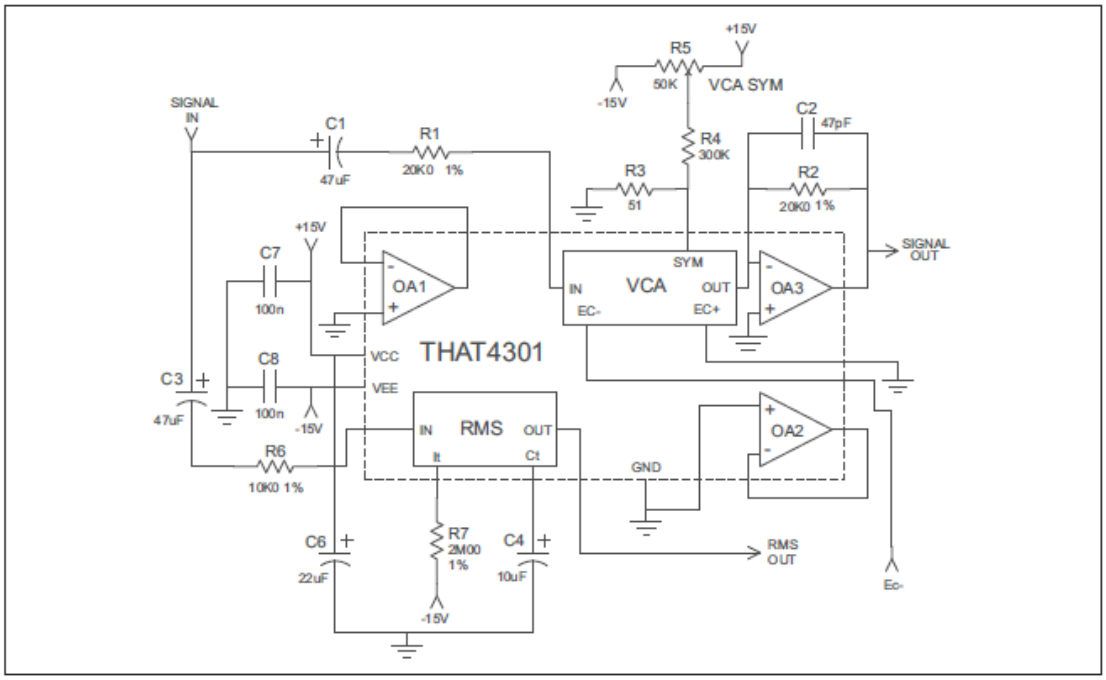


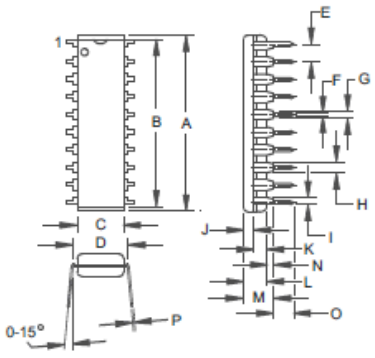
Figure 2. VCA and RMS detector test circuit

THAT Corporation; 45 Sumner Street; Milford, Massachusetts 01757-1656; USA
Tel: +1 (508) 478-9200; Fax: +1 (508) 478-0990; Web: www.thatcorp.com

Pin Name	DIP Pin	SO Pin
RMS In	1	3
I_T (I_{Time})	2	4
No Connection	3	5
RMS Out	4	6
C_T (C_{Time})	5	7
OA2 -In	6	9
OA2 Out	7	10
OA2 +In	8	11
GND	9	12
VEE	10	13
VCC	11	18
OA3 Out	12	19
VCA Out	13	20
SYM	14	22
E_{C+}	15	23

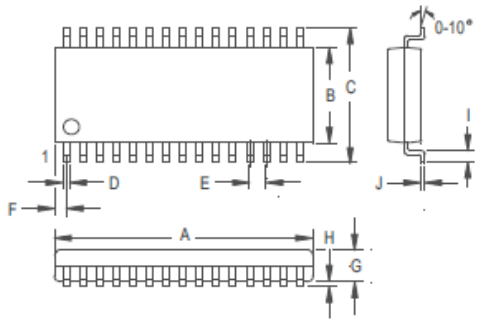
Pin Name	DIP Pin	SO Pin
E_{C-}	16	24
VCA In	17	25
OA1 Out	18	26
OA1 -In	19	27
OA1 +In	20	28
No Connection		1
No Connection		2
No Connection		8
No Connection		14
No Connection		15
No Connection		16
No Connection		17
No Connection		21
No Connection		29
No Connection		30

Table 2. Pin Connections



ITEM	MILLIMETERS	INCHES
A	24.8 Max.	0.98 Max
B	24.2 +/-0.2	0.95 +/-0.008
C	6.4 +/-0.2	0.25 +/-0.008
D	7.62 +/-0.25	0.30 +/-0.01
E	2.54 +/-0.15	0.10 +/-0.006
F	0.46 +0.15 -0.1	0.02 +0.006 -0.004
G	1.0 +/-0.15	0.04 +/-0.006
H	1.5 Typ.	0.06 Typ.
I	0.98 Typ.	0.04 Typ.
J	1.5	0.06
K	1.75	0.07
L	3.25 +/-0.15	0.13 +/-0.006
M	4.7 Max.	0.19 Max.
N	0.51 Min.	0.02 Min.
O	2.8 Min.	0.11 Min.
P	0.25 +0.15 -0.05	0.01 +0.006 -0.002

Figure 3. Plastic dual in-line package outline



ITEM	MILLIMETERS	INCHES
A	15.4 +/- 0.3	0.60 +/- 0.012
B	7.5 +/- 0.2	0.29 +/- 0.008
C	10.3 +/- 0.4	0.41 +/- 0.016
D	0.4 + 0.1 - 0.05	0.002 +0.004 -0.002
E	1.0 Typ.	0.039 Typ.
F	0.85 MAX.	0.033 Max.
G	2.3 +/- 0.15	0.09 +/- 0.006
H	0.15 +/- 0.1	0.006 +/- 0.004
I	0.8	0.031
J	0.2 + 0.1 - 0.05	0.008 +0.004 -0.002

Figure 4. Plastic surface-mount package outline

THAT Corporation; 45 Sumner Street; Milford, Massachusetts 01757-1656; USA
Tel: +1 (508) 478-9200; Fax: +1 (508) 478-0990; Web: www.thatcorp.com

Representative Data

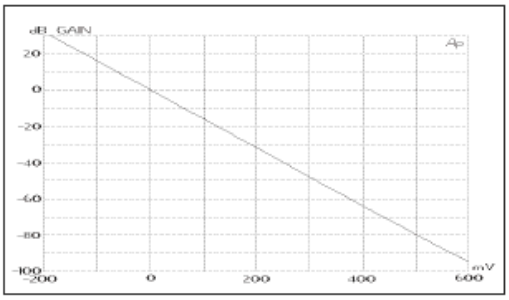


Figure 5. VCA Gain vs. Control Voltage (Ec-) at 25°C

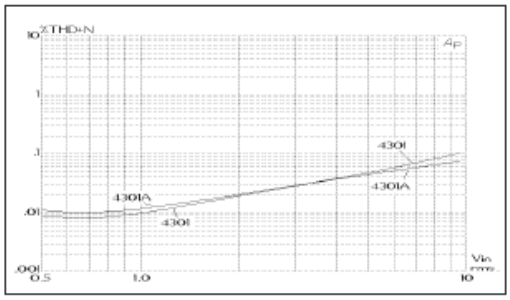


Figure 6. VCA 1kHz THD+Noise vs. Input, -15 dB Gain

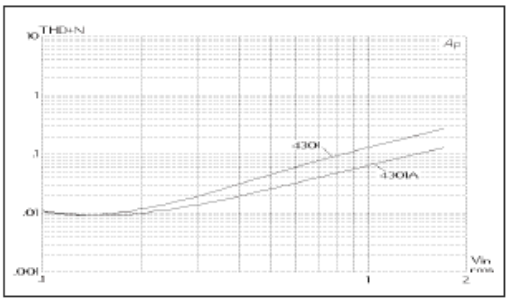


Figure 7. VCA 1kHz THD+Noise vs. Input, +15 dB Gain

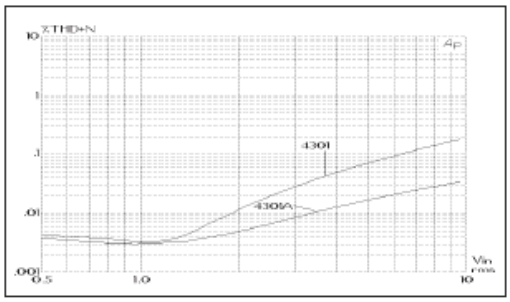


Figure 8. VCA 1kHz THD+Noise vs. Input, 0 dB Gain

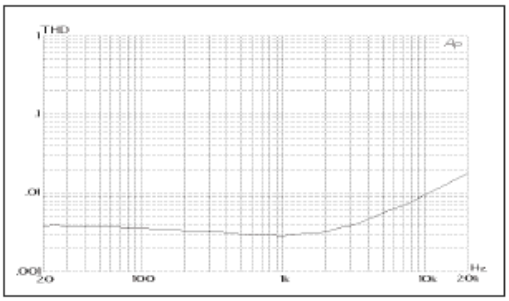


Figure 9. VCA THD vs. Frequency, 0 dB Gain, 1Vrms Input

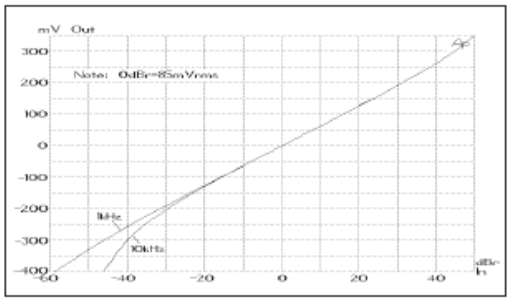


Figure 10. RMS Output vs. Input Level, 1 kHz & 10 kHz

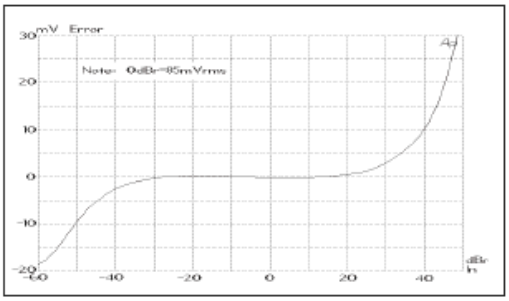


Figure 11. Departure from Ideal Detector Law vs. Level

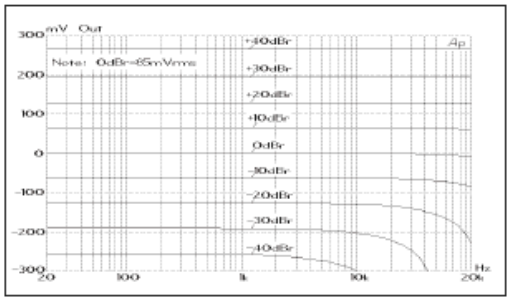
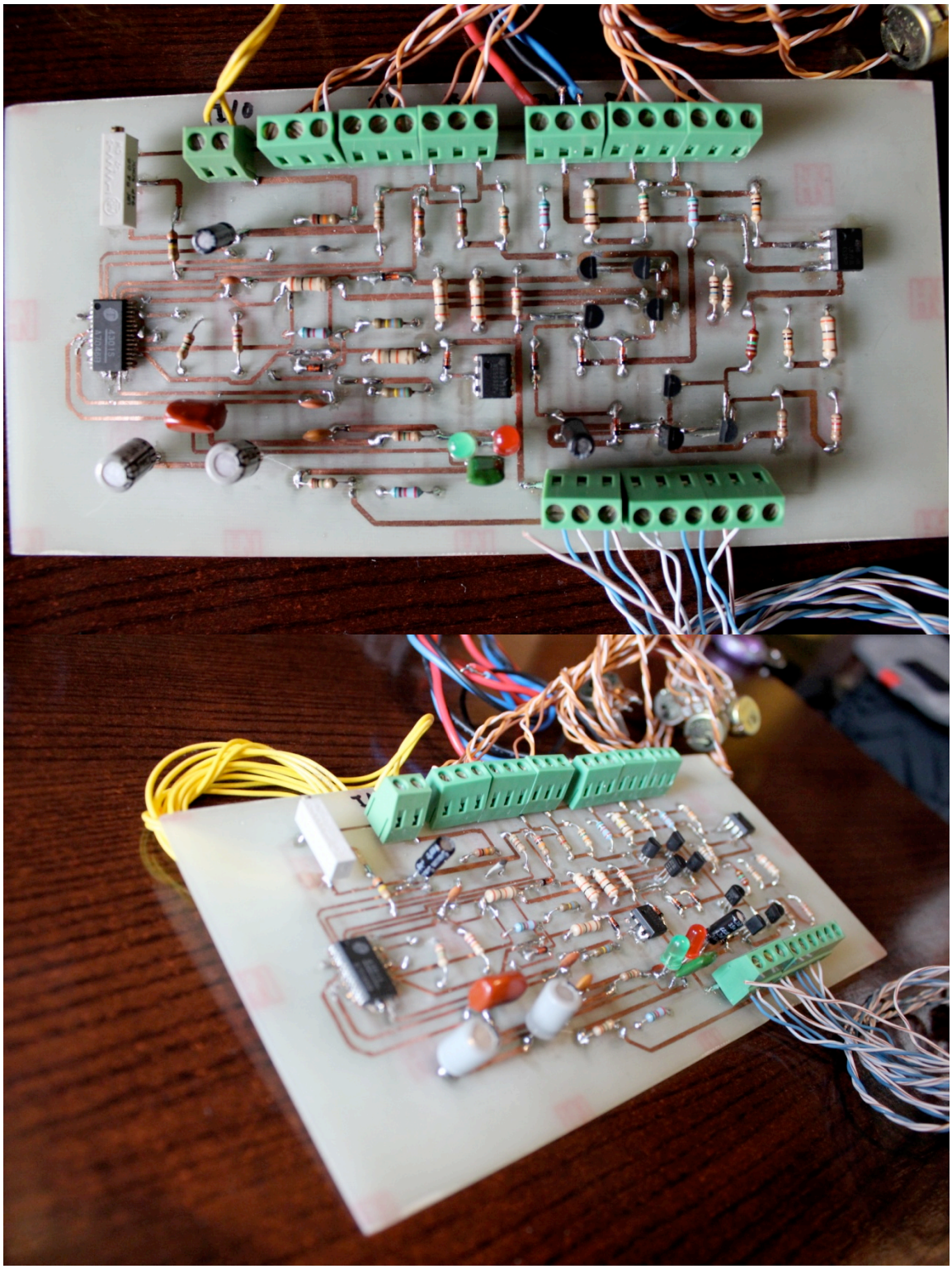
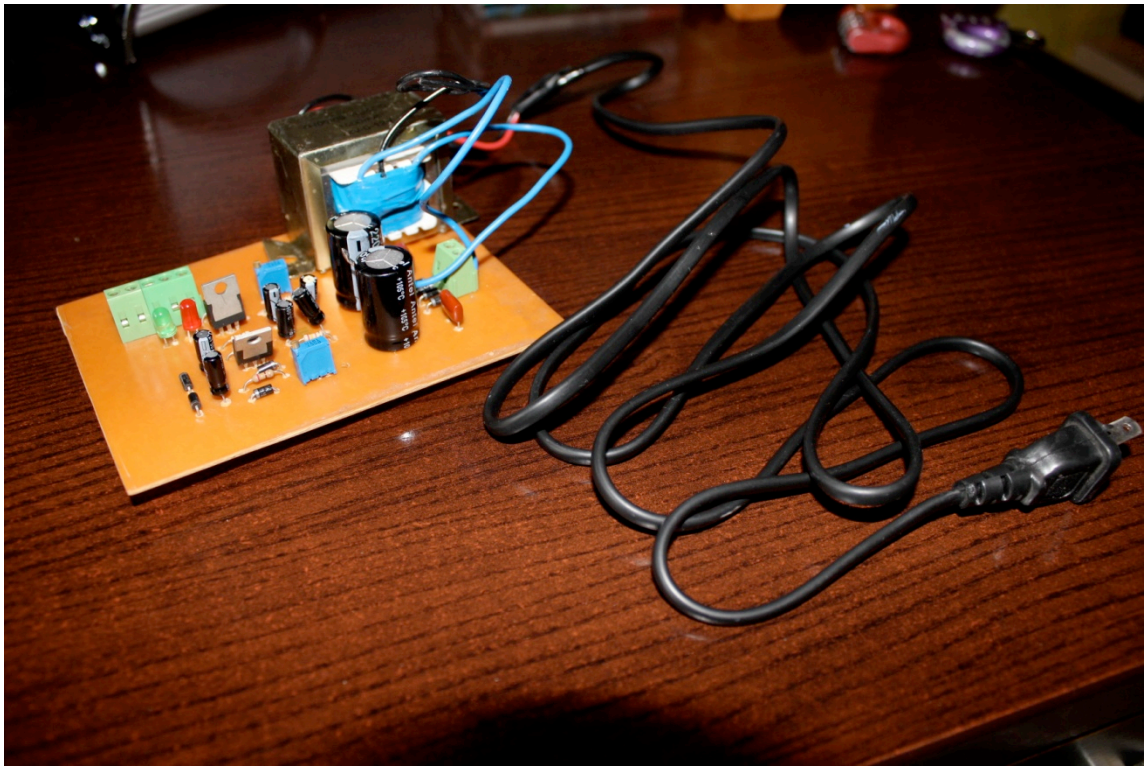


Figure 12. Detector Output vs. Frequency at Various Levels

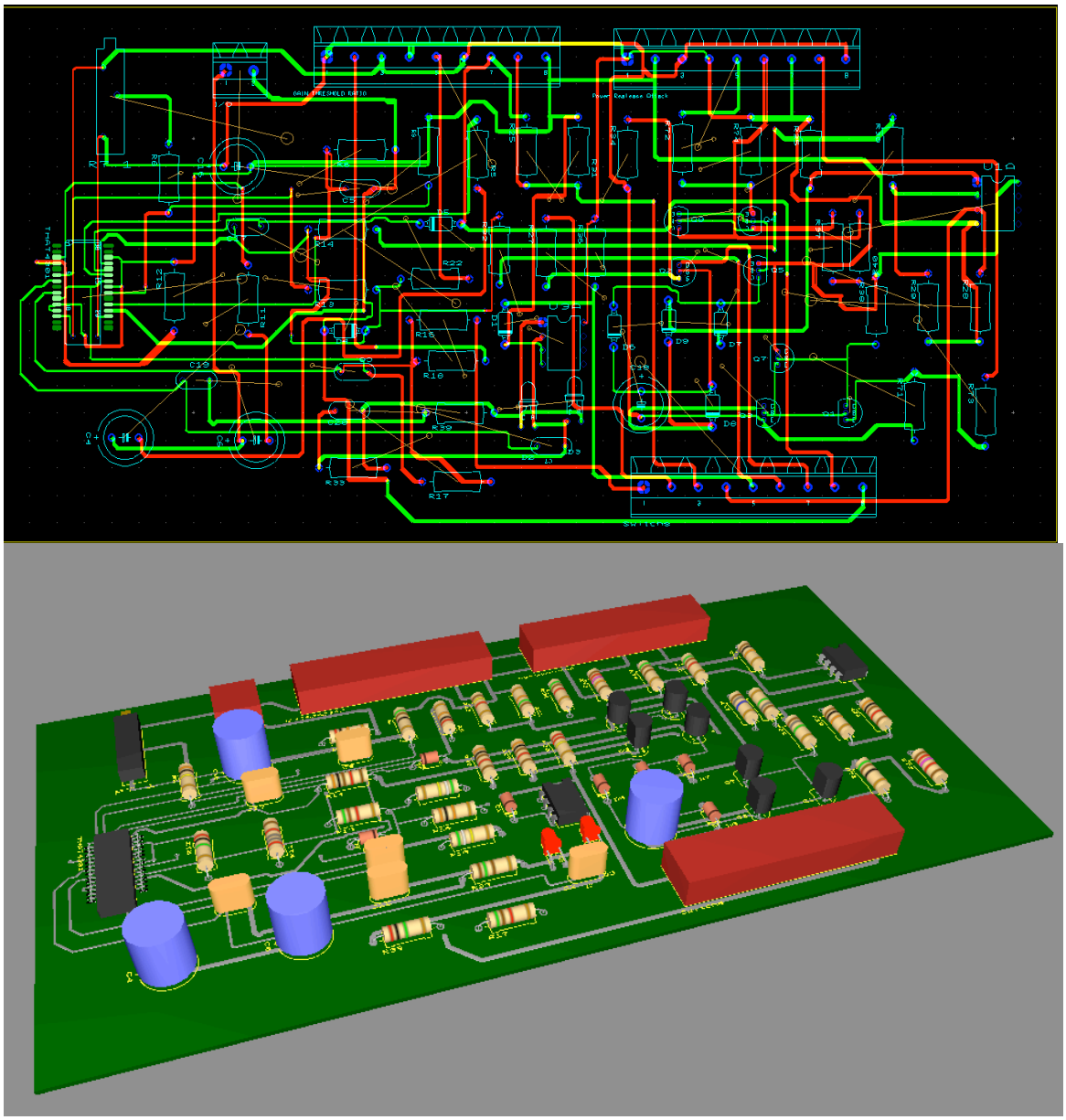
THAT Corporation; 45 Sumner Street; Milford, Massachusetts 01757-1656; USA
Tel: +1 (508) 478-9200; Fax: +1 (508) 478-0990; Web: www.thatcorp.com

Fotos de implementación analógica de un compresor





Layouts implementación analógica de un compresor



Anexos 2:

Archivos .M Matlab Usados:

Compresor

```
%=========================================================================%
% Compresor_final.m                                                       %
% Autor: Jose Luis Pinos                                                  %
% Tesis de grado para la obtencion del titulo de Ingeniero Electronico    %
% 14/09/2010                                                              %
% Implementacion de un compresor con parametros variables: attack time,   %
% realease time, threshhold y ratio.                                      %
% Implementacion de voice over compression con los mismos parametros de   %
% un compresor.                                                           %
%=========================================================================%



clear all;
close all;

% wavfile = 'Live.wav';
 wavfile = 'CBEDIT-DEADCAT.wav';
% wavfile = 'GEDIT-DEDCAT.wav';
% wavfile = 'GEDIT-JIMMY.wav';
% wavfile = 'KEDIT-DEADCAT.wav';
% wavfile = 'MEDIT-JIMMY.wav';
% wavfile = 'MEDIT-SMOKE.wav';
% wavfile = 'VEDIT-DEADCAT.wav';
% wavfile = 'VEDIT-JIMMY.wav';

wavfile2 = 'Sidechain.wav';

[x fs nbits] = wavread(wavfile); % x : vector que contiene el audio
                                 % fs : frecuencia de nuestreo
                                 % nbits : numero de bits


x = x';

%================================%
%    Parametros del compresor    %
%================================%

R = 5;              % Ratio
thdb = 40;           % Threshold en decibels
ATT = 10e-3;        % Attack time
RLS = 100e-3;         % Release time
TRT = .01e-3;       % Triggering time


%=====================================%
%    Calculo de constantes de tiempo   %
%=====================================%


T = 1/44100;                % Periodo de muestreo T.
AT = 1-exp(-2.2*T/ATT);  % Attack time para el periodo de muestreo T.
RT = 1-exp(-2.2*T/RLS);  % Release time para el periodo de muestreo T.
TR = 1-exp(-2.2*T/TRT);  % Triggering time para el periodo de muestreo T.

%=========================%
%    Ajuste threshold      %
```

```
%==========================%


th1=10^(-thdb/20); % Ajuste logaritmico de threshold


th = 10^(-thdb/20)*0.3; % Ajuste logaritmico de threshold


%==========================%
%    Ajuste de ganancia    %
%==========================%


%=========== Calculo de la ganancia estatica y dinamica ===============%

% Ganancia estatica: Se deriva de la grafica de la funcion de
% transferencia de un compresor.
% La formula de esta grafica es: Y = TH + (X - TH) / R.
% En el dominio logaritmico la funcion de transferencia equivale a la
% resta de Y - X.  De donde derivamos que la funci?n de control del
% compresor en dominio logaritmico es SG = (X - TH) / R + TH - X.

% Ganancia dinamica:  Se deriva de la ecuacion a diferencias del grafico
% del libro de Zolzer.
% La ecuacion es: Y[n] = AT * SG[n] + Y[n-1] * (1-AT) para attack time
% y, Y[n] = RT * SG[n] + Y[n-1] * (1-RT) para release time.

logrms=zeros(size(x));      % Inicializa vector logrms.
static_gain=ones(size(x));  % Inicializa vector static_gain.
dynamic_gain=ones(size(x)); % Inicializa vector dynamic_gain.

for n=2:length(x)
    % Calculo RMS de la senal
    logrms(n) = (TR*(2*log10(abs(x(n))+1))+logrms(n-1)*(1-TR))/2;
    if logrms(n)>=th
        % Ganancia estatica
        %static_gain(n)=(th/logrms(n))+((logrms(n)-th)/(R*logrms(n)));
        static_gain(n)=10^(((logrms(n)-th)/R+th-logrms(n)))*0.3;
        % Ganancia dinamica en attack time
        dynamic_gain(n)=AT*static_gain(n)+(1-AT)*dynamic_gain(n-1);
    else
        % Ganancia dinamica en release time
        dynamic_gain(n)=RT*static_gain(n)+(1-RT)*dynamic_gain(n-1) ;
    end;
end;

%============== Calculo de salida con ajuste ganancia ==============%

y = x.*dynamic_gain; % Salida con ajuste de ganancia


%================================================%
%    Graficas/resultados del compresor           %
%================================================%

time=linspace(0,length(x)/fs,length(x)); % Vector para graficar
                                         % en el tiempo

%=========== Figura 1 muestra entrada y resultados finales ============%

figure(1)

subplot(2,1,1)
hold on;
plot(time,x)
plot(time,th1*ones(size(x)), 'r');
axis([0 max(time) -1.1 1.1])
title('Input')
%xlabel('Segs')
```

```
grid on;
hold off;

subplot(2,1,2)
hold on;
plot(time,logrms)
plot(time,th*ones(size(x)), 'r');
axis([0 max(time) -0.1 0.35])
title('Senal RMS en dominio logaritmico')
%xlabel('Segs')
grid on;
hold off;

figure(2)

subplot(3,1,1)
hold on
plot(time,static_gain)
axis([0 max(time) -1.1 2.1])
title('Linear Static Gain')
%xlabel('Segs')
grid on
hold off

subplot(3,1,2)
hold on;
plot(time,dynamic_gain)
axis([0 max(time) -0.1 2.1])
title('Linear Dynamic Gain')
%xlabel('Segs')
grid on;
hold off;

subplot(3,1,3)
hold on;
plot(time,y)
plot(time,th1*ones(size(x)), 'r');
axis([0 max(time) -1.1 1.1])
title('Compressor output')
xlabel('Segs')
grid on;
hold off;


%
% %==============%
% %    EXTRA     %
% %==============%
%
% % Ciclo que prueba que los valores negativos del vector PCM no representan
% % las muestras de menor intensidad.
%
% xneg=zeros(size(x));
%
%
% for n=1:length(x);
%     if x(n)<0
%         xneg(n)=x(n);
%     end
% end
%
% %====================================%
% %    Escritura de archivos wave      %
% %====================================%
%
% wavwrite(xneg,fs,nbits,'PCM neg');    % Archivo wave que prueba que los
%                                       % valores negativos del vector PCM
%                                       % no representan las muestras de
%                                       % menor intensidad.
```

```
 wavwrite(y,fs,nbits,'CEdit Dead Cat Prueba 1');  % Archivo wave de salida
```

Compresor con Side Chain

```
%=========================================================================%
% Compresor_finalSC.m                                                     %
% Autor: Jose Luis Pinos                                                  %
% Tesis de grado para la obtencion del titulo de Ingeniero Electronico    %
% 14/09/2010                                                              %
% Implementacion de un compresor con parametros variables: attack time,   %
% realease time, threshhold y ratio.                                      %
% Implementacion de voice over compression con los mismos parametros de   %
% un compresor.                                                           %
%=========================================================================%



clear all;
close all;

% wavfile = 'Live.wav';
% wavfile = 'CBEDIT-DEADCAT.wav';
% wavfile = 'GEDIT-DEDCAT.wav';
% wavfile = 'GEDIT-JIMMY.wav';
% wavfile = 'KEDIT-DEADCAT.wav';
% wavfile = 'MEDIT-JIMMY.wav';
 wavfile = 'MEDIT-SMOKE.wav';
 wavfile2 = 'VEDIT-DEADCAT.wav';
% wavfile = 'VEDIT-JIMMY.wav';


[x fs nbits] = wavread(wavfile); % x : vector que contiene el audio
                                 % fs : frecuencia de nuestreo
                                 % nbits : numero de bits

[x2 fs2 nbits2] = wavread(wavfile2); % x : vector que contiene el audio
                                     % fs : frecuencia de nuestreo
                                     % nbits : numero de bits

x = x';
x2 = x2';
x2 = [x2 zeros(1,length(x)-length(x2))];



%================================%
%    Parametros del compresor    %
%================================%

R = 10;              % Ratio
thdb = 15;           % Threshold en decibels
ATT = 10e-3;        % Attack time
RLS = 1;         % Release time
TRT = .01e-3;       % Triggering time


%=======================================%
%    Calculo de constantes de tiempo    %
%=======================================%


T = 1/44100;                % Periodo de muestreo T.
AT = 1-exp(-2.2*T/ATT);  % Attack time para el periodo de muestreo T.
RT = 1-exp(-2.2*T/RLS);  % Release time para el periodo de muestreo T.
TR = 1-exp(-2.2*T/TRT);  % Triggering time para el periodo de muestreo T.

%=========================%
%    Ajuste threshold     %
```

```
%==========================%


th1=10^(-thdb/20); % Ajuste logaritmico de threshold


th = 10^(-thdb/20)*0.3; % Ajuste logaritmico de threshold


%==========================%
%    Ajuste de ganancia    %
%==========================%


%=========== Calculo de la ganancia estatica y dinamica ===============%

% Ganancia estatica: Se deriva de la grafica de la funcion de
% transferencia de un compresor.
% La formula de esta grafica es: Y = TH + (X - TH) / R.
% En el dominio logaritmico la funcion de transferencia equivale a la
% resta de Y - X.  De donde derivamos que la funci?n de control del
% compresor en dominio logar?tmico es SG = (X - TH) / R + TH - X.

% Ganancia dinamica:  Se deriva de la ecuacion a diferencias del grafico
% del libro de Zolzer.
% La ecuacion es: Y[n] = AT * SG[n] + Y[n-1] * (1-AT) para attack time
% y, Y[n] = RT * SG[n] + Y[n-1] * (1-RT) para release time.

logrms=zeros(size(x));      % Inicializa vector logrms.
static_gain=ones(size(x));  % Inicializa vector static_gain.
dynamic_gain=ones(size(x)); % Inicializa vector dynamic_gain.

for n=2:length(x2)
    % Calculo RMS de la senal
    logrms(n) = (TR*(2*log10(abs(x2(n))+1))+logrms(n-1)*(1-TR))/2;
    if logrms(n)>=th
        % Ganancia estatica
        %static_gain(n)=(th/logrms(n))+((logrms(n)-th)/(R*logrms(n)));
        static_gain(n)=10^(((logrms(n)-th)/R+th-logrms(n)))*0.3;
        % Ganancia dinamica en attack time
        dynamic_gain(n)=AT*static_gain(n)+(1-AT)*dynamic_gain(n-1);
    else
        % Ganancia dinamica en release time
        dynamic_gain(n)=RT*static_gain(n)+(1-RT)*dynamic_gain(n-1) ;
    end;
end;

%============== Calculo de salida con ajuste ganancia ==============%

y = x.*dynamic_gain; % Salida con ajuste de ganancia

%================================================%
%    Graficas/resultados del compresor           %
%================================================%

time=linspace(0,length(x)/fs,length(x)); % Vector para graficar
                                         % en el tiempo

%=========== Figura 1 muestra entrada y resultados finales ============%

figure(1)

subplot(2,1,1)
hold on;
plot(time,x2)
plot(time,th1*ones(size(x)), 'r');
axis([0 max(time) -1.1 1.1])
title('Side Chain Input')
%xlabel('Segs')
grid on;
```

```
hold off;

subplot(2,1,2)
hold on;
plot(time,logrms)
plot(time,th*ones(size(x)), 'r');
axis([0 max(time) -0.1 0.35])
title('Senal RMS en dominio logaritmico')
%xlabel('Segs')
grid on;
hold off;

figure(2)

subplot(3,1,1)
hold on
plot(time,static_gain)
axis([0 max(time) -1.1 2.1])
title('Static Gain')
%xlabel('Segs')
grid on
hold off

subplot(3,1,2)
hold on;
plot(time,dynamic_gain)
axis([0 max(time) -0.1 2.1])
title('Dynamic Gain')
%xlabel('Segs')
grid on;
hold off;

subplot(3,1,3)
hold on;
plot(time,y)
plot(time,th1*ones(size(x)), 'r');
axis([0 max(time) -1.1 1.1])
title('Salida comprimida por senal de Side Chain')
xlabel('Segs')
grid on;
hold off;

%====================================%
%    Escritura de archivos wave      %
%====================================%

 wavwrite(y,fs,nbits,'CEdit Dead Cat Prueba 1');  % Archivo wave de salida
```

Compuerta de ruido

```
%=========================================================================%
% Noisegate_final.m                                                       %
% Autor: Jose Luis Pinos                                                  %
% Tesis de grado para la obtencion del titulo de Ingeniero Electronico    %
% 14/09/2010                                                              %
% Implementacion de un noise gate con parametros variables: attack time, %
% realease time, hold time e histeresis (thresholh superior e inferior). %
% Implementacion de modo ducker con los mismos parametros del noise gate.%
%=========================================================================%



clear all;
close all;

wavfile = 'live.wav';
```

```
[x fs nbits] = wavread(wavfile); % x : vector que contiene el audio
                                 % fs : frecuencia de nuestreo
                                 % nbits : numero de bits

x = x';

%==================================%
%    Parametros del noise gate     %
%==================================%

thresh_up=10;                     % Threshold superior
hysteresis=10;                    % Hysteresis
thresh_dwn=hysteresis+thresh_up;  % Threshold inferior
ATTACK = 300e-6;                  % Attack time
RELEASE = 100e-3;                 % Release time
HOLD = 10e-3;                     % Hold time

%===========================%
%    Ajuste threshold       %
%===========================%

thup1=10^(-thresh_up/20); % Ajuste logaritmico de threshold


thup = 10^(-thresh_up/20)*0.3; % Ajuste logaritmico de threshold

thdwn1=10^(-thresh_dwn/20); % Ajuste logaritmico de threshold


thdwn = 10^(-thresh_dwn/20)*0.3; % Ajuste logaritmico de threshold


%================================================================%
%    Inicializa parametros para calculo PEAK de la senal         %
%================================================================%

peak = zeros(size(x));       % Inicializa vector peak
peak2 = zeros(size(x));      % Inicializa vector peak

TRA=0.01e-4;     % Peakmeter attack time
TRR=0.1;          % Peakmeter release time


T=1/fs;                          % Periodo de muestreo T.

TRIG_ATT =  1-exp(-2.2*T./TRA); % Peakmeter attack time para el periodo
                                 % de muestreo T.
TRIG_RLS =  1-exp(-2.2*T./TRR); % Peakmeter release time para el periodo
                                 % de muestreo T.

%===========================%
%    Ajuste de ganancia     %
%===========================%

%================== Calculo de la ganancia dinamica ==================%


nrel=round(RELEASE*fs); %number of samples for fade
natt=round(ATTACK*fs); %number of samples for fade
nhd=round(HOLD*fs);
contrelease=0;
contattack=0;
conthold=0;
verifgate=0;
verifrel=0;
finattack=0;
dynamic_gain=zeros(size(x)); % Inicializa vector static_gain.
peakdB=zeros(size(x));
```

```
peakdB2=zeros(size(x));

for i=2:length(x);
    n=i;
    z = abs(x(n))-peak(n-1); % Se detecta si el nivel incrementa
    if z < 0
        z = 0;
    end;
    peak(n) = z.*TRIG_ATT + peak(n-1)*(1 - TRIG_RLS);
    peakdB(n) = log10(peak(n)+1);
    if ((peakdB(i)>=thup) && finattack==0) && (contattack < natt)
        contattack=contattack+1;
        verifgate=1;
        dynamic_gain(i)=dynamic_gain(i-1)+1/natt;
        if contattack==natt
            finattack=1;
            verifrel=0;
        end
    elseif ((finattack==1) && ((conthold < nhd) || (peakdB(i)>=thdwn))) &&
verifrel==0
         conthold=conthold+1;
         dynamic_gain(i)=1;
    else
        verifrel=1;
        if verifgate==1
            contrelease=contrelease+1;
            dynamic_gain(i)=dynamic_gain(i-1)-1/nrel;
            if contrelease==nrel
                finattack=0;
                verifgate=0;
                contrelease=0;
                conthold=0;
                contattack=0;
                verifrel=0;
            end
        end
    end
end


%============== Calculo de salida con ajuste ganancia ==============%

y = x.*dynamic_gain; % Salida con ajuste de ganancia


%================================================%
%    Graficas/resultados del noise gate        %
%================================================%

time=linspace(0,length(x)/fs,length(x)); % Vector para graficar
                                         % en el tiempo

%=========== Figura 1 muestra entrada y resultados finales ============%

figure(1)

subplot(2,1,1)
hold on;
plot(time,x)
plot(time,thup1*ones(size(x)), 'r');
plot(time,thdwn1*ones(size(x)), 'g');
axis([0 max(time) -1.1 1.1])
title('Input')
%xlabel('Segs')
grid on;
hold off;

subplot(2,1,2)
hold on;
plot(time,peakdB)
```

```
plot(time,thup*ones(size(x)), 'r');
plot(time,thdwn*ones(size(x)), 'g');
axis([0 max(time) -0.1 0.35])
title('Senal PEAK en dominio logaritmico')
%xlabel('Segs')
grid on;
hold off;

figure(2)

subplot(2,1,1)
hold on;
plot(time,dynamic_gain)
axis([0 max(time) -0.1 1.1])
title('Dynamic Gain')
%xlabel('Segs')
grid on;
hold off;

subplot(2,1,2)
hold on;
plot(time,y)
axis([0 max(time) -1.1 1.1])
title('Noise Gate output')
xlabel('Segs')
grid on;
hold off;


%====================================%
%    Escritura de archivo wave       %
%====================================%

wavwrite(y,fs,nbits,'Gated signal');  % Archivo wave de salida
```

Expansor

```
%============================================================================%
% Expander_final.m                                                           %
% Autor: Jose Luis Pinos                                                     %
% Tesis de grado para la obtencion del titulo de Ingeniero Electronico       %
% 14/09/2010                                                                 %
% Implementacion de un expansor con parametros variables: attack time,       %
% realease time, threshhold y ratio.                                         %
%============================================================================%

% Comentarios: Expander no debe tener un attack time muy elevado porque
% atenua la senal, ya que al atenuar muy rapido se atenuan tambien los
% transitorios de alta intensidad. Buen attack time: 50ms

clear all;
close all;

wavfile = 'live.wav';


[x fs nbits] = wavread(wavfile); % x : vector que contiene el audio
                                 % fs : frecuencia de nuestreo
                                 % nbits : numero de bits

x = x';


%================================%
%    Parametros del expansor     %
%================================%
```

```
R = 20;              % Ratio expander
thdb = 5;            % Threshold en decibels
ATT = 50e-3;         % Attack time
RLS = 100e-3;        % Release time
TRT = 0.01e-3;       % Triggering time



%======================================%
%    Calculo de constantes de tiempo   %
%======================================%


T = 1/fs;                % Periodo de muestreo T.
AT = 1-exp(-2.2*T./ATT);  % Attack time para el periodo de muestreo T.
RT = 1-exp(-2.2*T./RLS);  % Release time para el periodo de muestreo T.
TR = 1-exp(-2.2*T./TRT);  % Triggering time para el periodo de muestreo T.

%============================%
%    Ajuste threshold      %
%============================%

th1=10^(-thdb/20); % Ajuste logaritmico de threshold


th = 10^(-thdb/20)*0.3; % Ajuste logaritmico de threshold

Range=th/(R+1);


%=========== Calculo de la ganancia estatica y dinamica ===============%

% Ganancia estatica: Se deriva de la grafica de la funcion de
% transferencia de un expansro.
% La formula de esta grafica es: Y = (TH - X) / R.
% En el dominio logaritmico la funcion de transferencia equivale a la
% resta de Y - X.  De donde derivamos que la funcion de control del
% expansor en dominio logaritmico es SG = (TH - X) / R - X.

% Ganancia dinamica:  Se deriva de la ecuacion a diferencias del grafico
% del libro de Zolzer.
% La ecuacion es: Y[n] = AT * SG[n] + Y[n-1] * (1-AT) para attack time
% y, Y[n] = RT * SG[n] + Y[n-1] * (1-RT) para release time.

logrms=zeros(size(x));           % Inicializa vector logrms.
static_gain=ones(size(logrms));  % Inicializa vector static_gain.
dynamic_gain=ones(size(logrms)); % Inicializa vector dynamic_gain.


for n=2:length(x)
    % Calculo RMS de la senal
    logrms(n) = (TR*(2*log10(abs(x(n))+1))+logrms(n-1)*(1-TR))/2;
    if logrms(n)<=th && logrms(n)>=Range
        % Ganancia estatica en attack time
        static_gain(n)=10^((th-logrms(n))/R-logrms(n))*0.3;
        % Ganancia dinamica en attack time
        dynamic_gain(n)=AT*static_gain(n)+(1-AT)*dynamic_gain(n-1);
    else
        % Ganancia dinamica en release time
        dynamic_gain(n)=RT*static_gain(n)+(1-RT)*dynamic_gain(n-1) ;
    end;
end;

%============== Calculo de salida con ajuste ganancia ==============%

y = x.*dynamic_gain; % Salida con ajuste de ganancia
```

```
%=============================================%
%    Graficas/resultados del expansor         %
%=============================================%

time=linspace(0,length(x)/fs,length(x)); % Vector para graficar
                                         % en el tiempo

%=========== Figura 1 muestra entrada y resultados finales ============%

figure(1)

subplot(2,1,1)
hold on;
plot(time,x)
plot(time,th1*ones(size(x)), 'r');
axis([0 max(time) -1.1 1.1])
title('Input')
%xlabel('Segs')
grid on;
hold off;

subplot(2,1,2)
hold on;
plot(time,logrms)
plot(time,th*ones(size(x)), 'r');
axis([0 max(time) -0.1 0.35])
title('Senal RMS en dominio logaritmico')
%xlabel('Segs')
grid on;
hold off;

figure(2)

subplot(3,1,1)
hold on
plot(time,static_gain)
axis([0 max(time) -0.1 1.1])
title('Linear Static Gain')
%xlabel('Segs')
grid on
hold off

subplot(3,1,2)
hold on;
plot(time,dynamic_gain)
axis([0 max(time) -0.1 1.1])
title('Linear Dynamic Gain')
%xlabel('Segs')
grid on;
hold off;

subplot(3,1,3)
hold on;
plot(time,y)
plot(time,th1*ones(size(x)), 'r');
axis([0 max(time) -1.1 1.1])
title('Expander output')
xlabel('Segs')
grid on;
hold off;

%====================================%
%    Escritura de archivo wave       %
%====================================%

wavwrite(y,fs,nbits,'Expanded signal');  % Archivo wave de salida
```

Limitador

```
%=========================================================================%
% Limiter_final.m                                                         %
% Autor: Jose Luis Pinos                                                  %
% Tesis de grado para la obtencion del titulo de Ingeniero Electronico    %
% 14/09/2010                                                              %
% Implementacion de un limitador con parametros variables: attack time,   %
% realease time y threshhold.                                             %
%=========================================================================%

clear all;
close all;

wavfile = 'Live.wav';

[x fs nbits] = wavread(wavfile); % x : vector que contiene el audio
                                 % fs : frecuencia de nuestreo
                                 % nbits : numero de bits

x = x';

%================================%
%    Parametros del limitador    %
%================================%

thdb = 20;        % Threshold en decibels
TRA=0.01e-3;      % Peakmeter attack time
TRR=1e-3;         % Peakmeter release time
ATT=1e-3;       % Attack time
RLS=100e-3;          % Release time

%=====================================%
%    Calculo de constantes de tiempo  %
%=====================================%

T = 1/fs;                          % Periodo de muestreo T.
TRIG_ATT =  1-exp(-2.2*T./TRA); % Peakmeter attack time para el periodo
                                   % de muestreo T.
TRIG_RLS =  1-exp(-2.2*T./TRR); % Peakmeter release time para el periodo
                                   % de muestreo T.
AT = 1-exp(-2.2*T./ATT);           % Attack time para el periodo de
                                   % muestreo T.
RT = 1-exp(-2.2*T./RLS);           % Release time para el periodo de
                                   % muestreo T.


peak = zeros(size(x));        % Inicializa vector peak
peakdB = zeros(size(x));      % Inicializa vector peakdB
static_gain=ones(size(x));  % Inicializa vector static_gain.
dynamic_gain=ones(size(x)); % Inicializa vector dynamic_gain.

%===========================%
%    Ajuste threshold       %
%===========================%

th1=10^(-thdb/20); % Ajuste logaritmico de threshold


th = 10^(-thdb/20)*0.3; % Ajuste logaritmico de threshold


%===========================%
%    Ajuste de ganancia     %
%===========================%

%================== Calculo de la ganancia estatica ==================%
```

```
% Se deriva de la grafica de la funcion de transferencia de un compresor.
% La formula de esta grafica es: Y / X = TH / X + (X - TH) / R*X.
% En el dominio logaritmico la funcion de transferencia equivale a la
% resta de Y - X.  De donde derivamos que la funcion de control del
% compresor en dominio logaritmico es SG = (X - TH) / R + TH - X.
% Sin embargo para un limitador R es infinito por lo tanto SG = TH - X.



%================== Calculo de la ganancia dinamica ==================%

% Se deriva de la ecuaci?n a diferencias del grafico del libro de Zolzer.
% La ecuacion es: Y[n] = AT * SG[n] + Y[n-1] * (1-AT) para attack time
% y, Y[n] = RT * SG[n] + Y[n-1] * (1-RT) para release time.


for n=2:length(x);
    z = abs(x(n))-peak(n-1); % Se detecta si el nivel incrementa
    if z < 0
        z = 0;
    end;
    peak(n) = z.*TRIG_ATT + peak(n-1)*(1 - TRIG_RLS);
    peakdB(n) = log10(peak(n)+1);
    if peakdB(n)>=th
        static_gain(n)=10^(th - peakdB(n))*0.3;
        dynamic_gain(n)=AT*static_gain(n)+(1-AT)*dynamic_gain(n-1);
    else
        dynamic_gain(n)=RT*static_gain(n)+(1-RT)*dynamic_gain(n-1) ;
    end
end

%============== Calculo de salida con ajuste ganancia ==============%

y = x.*dynamic_gain; % Salida con ajuste de ganancia

%================================================%
%    Graficas/resultados del limitador          %
%================================================%

time=linspace(0,length(x)/fs,length(x)); % Vector para graficar
                                         % en el tiempo


figure(1)

subplot(2,1,1)
hold on;
plot(time,x)
plot(time,th1*ones(size(x)), 'r');
axis([0 max(time) -1.1 1.1])
title('Input')
%xlabel('Segs')
grid on;
hold off;

subplot(2,1,2)
hold on;
plot(time,peak)
plot(time,th1*ones(size(x)), 'r');
axis([0 max(time) -0.1 1.1])
title('Senal PEAK')
%xlabel('Segs')
grid on;
hold off;

figure(2)

subplot(3,1,1)
hold on
plot(time,static_gain)
```

```
axis([0 max(time) -0.1 2.1])
title('Static Gain')
%xlabel('Segs')
grid on
hold off

subplot(3,1,2)
hold on;
plot(time,dynamic_gain)
axis([0 max(time) -0.1 2.1])
title('Dynamic Gain')
%xlabel('Segs')
grid on;
hold off;

subplot(3,1,3)
hold on;
plot(time,y)
plot(time,th1*ones(size(x)), 'r');
axis([0 max(time) -1.1 1.1])
title('Limiter output')
xlabel('Segs')
grid on;
hold off;

%====================================%
%    Escritura de archivo wave       %
%====================================%

wavwrite(y,fs,nbits,'Limited signal');  % Archivo wave de salida
```