Algoritmo de síntesis para el método de logaritmo de seno.

Principio básico.

El algoritmo empleado en el sintetizador de audio está inspirado en el usado en los chips OPL3 de Yamaha (YMF262), sin embargo se encuentra optimizado y adaptado para funcionar en los microcontroladores AVR de Arduino, empleando aritmética de punto fijo de 16 y 24 bits, así como tablas de 8 bits tanto en direcciones (256 entradas) como en valores almacenados (rango dinámico de 256 valores distintos). Esto crea un sonido de una calidad que técnicamente es baja, pero con resultados de una calidad sorprendentemente aceptable y muy superior a lo que se puede lograr mediante la función tone de Arduino.

El esquema de modulación utilizado es el de modulación de amplitud (AM), el cual imita con aceptable fidelidad el comportamiento de los instrumentos reales, los cuales tienden a mantener la misma frecuencia de oscilación a lo largo de las notas, pero con cambios en amplitud conforme avanza el tiempo, presentando una disminución exponencial inversa. Este esquema de modulación se plantea mediante la siguiente ecuación:

$$V(t)=A\cdot Sen(\omega t)$$

Donde V(t) es la tensión de salida en función del tiempo (que se corresponde directamente con la intensidad instantánea de la onda acústica), A es la amplitud de la onda sinusoidal (que esta asociada al volumen o intensidad del sonido) y ω es la frecuencia angular de la onda sinusoidal (que determina el tono de cada nota).

En la implementación mediante Arduino, se usarán parámetros discretizados de tiempo y amplitud, por lo que definiremos una versión discreta de la fórmula para cada una de las muestras “m”:

$$m=a\cdot sen\,\alpha$$

Donde “a” representa la amplitud normalizada y “α” representa el ángulo discreto que se emplea como argumento de la función seno. Dado que se emplean 8 bits de resolución para estos ángulos (los cuales a su vez son calculados empleando aritmética de punto fijo), tenemos una gama de 2^8 = 256 valores posibles, determinados por la siguiente ecuación:

$$\alpha=\frac{n}{256}(2\pi),\ \ 0\leq n\leq 255$$

Donde “n” representa el numero de muestra (cada uno de los pasos discretos del ángulo), en una secuencia que va de 0 a 255 y luego comenzando desde 0

nuevamente, creando ángulos con separación uniforme entre 0 y 2π radianes.

La implementación directa de la modulación en amplitud requiere por tanto la multiplicación de valores de amplitud (la cual debe variarse con el tiempo) por los valores generados por la función sinusoidal, sin embargo en Arduino no se cuenta con unidad de punto flotante por hardware y esta clase de cálculos requeriría más poder del procesamiento del que se tiene a la mano, de ahí que se emplean técnicas de discretización para operar todo con números enteros, así como el empleo de tablas precalculadas que eliminan los cálculos más complejos.

Las multiplicaciones en sí no presentan un problema de rendimiento, dado que el CPU del Arduino cuenta con una unidad de multiplicación y pueden emplearse técnicas de discretización para este fin. Sin embargo la generación de la envolvente exponencial requiere punto flotante, y es aquí donde este algoritmo se vuelve atractivo, pues al aplicar logaritmos las multiplicaciones se convierten en sumas y la envolvente exponencial surge como consecuencia natural al ir disminuyendo (restando) los exponentes de forma lineal.

Las identidades empleadas por el método se presentan a continuación:

$$x = b^{\log_b(x)}, \qquad \log_b(x \cdot y) = \log_b(x) + \log_b(y)$$

Nótese que de manera intencional se elige emplear una base de logaritmo “b” la cual no es convencional (a diferencia de 10 o *e*), incluso se elige no usar la base 2 de logaritmo como en los chips OPL3 (la base 2 permite hacer corrimientos de bits para multiplicar o dividir por potencias de 2). La necesidad de estas bases arbitrarias se abordará más adelante.

Luego, aplicando logaritmo natural a la ecuación discreta de modulación en amplitud se tiene:

$$m = a \cdot sen\,\alpha$$
$$\log_b(m) = \log_b(a \cdot sen\,\alpha)$$
$$\log_b(m) = \log_b(a) + \log_b(sen\,\alpha)$$

Sin embargo debemos hacer un alto aquí, y notar que el logaritmo no está definido dentro del conjunto de los números reales para los valores de 0 y números negativos, y como se puede apreciar la función seno genera valores en el intervalo [-1, 1], por lo que se recurre a un artificio que consiste en generar solamente valores positivos para la función seno aplicando valor absoluto. Esto causará una pérdida del signo, pero más adelante se hará un ajuste de signo cuando se evalúe la función exponencial.

En el caso del cero, se opta por usar un artificio que consiste en hacer un sutil y a la vez conveniente corrimiento de fase, el cual evita que la función seno devuelva

cero. Este artificio consiste en mover la fase del argumento medio conteo a la derecha, es decir, sumar 0.5 al valor de n, de manera que jamás se generan de forma exacta los ángulos 0 y π:

$$\alpha = \frac{n+0.5}{256}(2\pi),\ \ 0 \leq n \leq 255$$

Nótese que este corrimiento genera cruces por cero que son simétricos (la muestra antes del cero dista la misma amplitud que la muestra después del cero), consiguiendo que el ajuste de signo posterior produzca cruces consistentes que no generan distorsión.

No está demás aclarar que se evita generar una tabla de ¼ de ciclo (ángulo de 0 a π/2) como se hace en el chip OPL3 porque en Arduino reflejar la tabla implica contar hacia atrás cada dos cuartos de ciclo, lo cual no se puede implementar eficientemente porque se usan cálculos de punto fijo para hacer el remuestreo. Lamentablemente con esto se pierden 2 bit de resolución efectiva que de otra forma podrían ayudar a mejorar la calidad de audio.

Luego, reescribiendo con valores absolutos y tomando en cuenta que la amplitud "a" sólo puede ser positiva:

$$\begin{aligned} |m| &= a \cdot |sen\,\alpha| \\ \log_b |m| &= \log_b (a \cdot |sen\,\alpha|) \\ \log_b |m| &= \log_b a + \log_b |sen\,\alpha| \\ b^{\log_b |m|} &= b^{\log_b a + \log_b |sen\,\alpha|} \\ |m| &= b^{\log_b a + \log_b |sen\,\alpha|} \end{aligned}$$

Esto permite una implementación de la ecuación que se puede optimizar mediante tablas. En este caso se implementan 2 tablas, una que evalúe el término del logaritmo del seno y otra que evalué la función exponencial, ambas con base "b".

El término del logaritmo de "a" puede ser ajustado de escala en forma arbitraria, siempre que "a" se mantenga en el intervalo de]0, 1] (para evitar el cero dentro del logaritmo), de tal forma que se emplean números enteros directamente, evitando así una tabla de logaritmos para este factor. Dado que la progresión del logaritmo de "a" será lineal, tras ser aplicada a una función exponencial se conseguirá una envolvente amortiguada muy característica de los sistemas resonantes, dando una atenuación "natural" a los sonidos generados.

De ahora en adelante, nos referiremos a este logaritmo como “σ“ o factor de atenuación:

$$\sigma = \log_b a$$

Asimismo, definiremos con la letra “p” a la potencia a la que se elevará la base:

$$p = \log_b a + \log_b |sen\,\alpha|$$
$$p = \sigma + \log_b |sen\,\alpha|$$

Luego, la ecuación de potencia quedará así:

$$|m| = b^{\sigma + \log_b |sen\,\alpha|}$$
$$|m| = b^p$$

Cálculo y normalización de los valores de las tablas.

Tabla exponencial.

Una aproximación usada en el chip OPL3 consiste en cambiar los signos de los exponentes para causar atenuaciones (valores menores que 1) en vez de amplificaciones. Esta inversión de signo puede hacerse innecesaria para nuestro caso, empleando la siguiente reducción:

$$|m| = b^{-p} \;\rightarrow\; |m| = \frac{1}{b^p} \;\rightarrow\; |m| = \left(\frac{1}{b}\right)^p$$

Como puede verse, al emplear un recíproco como base se obtiene el mismo comportamiento de atenuación que si el exponente fuera negativo. Por tanto, en nuestro caso simplemente emplearemos una base inferior a la unidad.

Lo anterior significa que no cambiaremos los signos de los exponentes, ya que gracias a esta propiedad podemos mantenerlos siempre positivos. Asimismo, implica que para causar atenuaciones en la amplitud simplemente iremos ***incrementando de forma positiva*** los exponentes.

Luego viene el problema de normalizar la amplitud de la tabla exponencial al máximo valor deseado. Esto es extremadamente simple de lograr con la función exponencial, dado que el valor máximo se puede generar al elevar la base a la potencia cero, independientemente de la base empleada:

$$1 = b^0 \;\rightarrow\; |m_{max}| = a_{norm} \cdot b^0$$

Donde “a_{norm}” es la amplitud normalizada que es igual a la amplitud pico de las muestras.

Dado que la salida de la tabla de función exponencial es de 8 bits, el rango de salida puede estar entre 0 y 255, sin embargo se elige no usar estos límites y reducirlos, dado que debe reservarse algo de rango dinámico para la mezcla de voces (caso contrario podría ocurrir desbordamiento en la salida, lo cual crea distorsiones desagradables).

Por la forma en que se configura el timer del Arduino se obtienen 9 bits de resolución efectiva, los cuales permiten generar valores con signo que van de -256 a +255. Con esto en mente se elige una amplitud máxima de 63, que en teoría permite mezclar hasta 4 canales (±63*4 = ±252). Luego:

$$a_{norm} = 63 \;\rightarrow\; |m| = 63 \cdot b^{p}$$

No está demás señalar que en la práctica sólo fue posible mezclar 3 canales, limitados por la velocidad de procesador del Arduino.

El siguiente problema que surge es el de generar la atenuación máxima; esto deberá ocurrir para el máximo valor que puede introducirse como entrada de la tabla exponencial y dado que la entrada de la tabla es de 8 bit, la posición máxima es p = 255. Es de esperar que esta última posición contenga un cero (para atenuar totalmente la amplitud), sin embargo una propiedad fundamental de las funciones exponenciales es que jamás pueden generar el valor de 0, dado que implicaría elevar a una potencia infinita. Esto se solventa mediante un artificio de redondeo, dado que si bien el valor de |m| va siempre en decremento, deberá llegar eventualmente a ser menor que 0.5, lo cual causaría que |m| se redondee hacia abajo, convirtiéndose en 0.

Por tanto, la base de la potencia se ajusta de tal forma que justo en la última posición de la tabla (p = 255) se genere un valor de 0.5. Si se excede el límite de la tabla (p > 255) simplemente se hace |m| igual a cero, ya que de todas formas serán valores menores a 0.5 y se convertirían a 0 por redondeo. No está demás señalar que valores de p mayores a 255 son normales en la realización del algoritmo y son manejados de esta forma.

Luego, despejando para la base:

$$\begin{aligned}
|m_{min}| &= a_{norm} \cdot b^{p_{max}} \\
0.5 &= 63\, b^{255} \\
\frac{0.5}{63} &= b^{255} \\
\sqrt[255]{\frac{0.5}{63}} &= b \\
b &= 0.981212908
\end{aligned}$$

Ya con la base obtenida, se puede escribir la expresión para la tabla exponencial:

$$|m|=63(0.981212908)^{p} \quad , \quad 0\leq p\leq 255$$

Cabe aclarar que los valores de |m| deberán ser redondeados al entero más cercano para que puedan ser almacenados en una tabla de enteros de 8 bits.

Tabla de logaritmo de seno.

El proceso de normalización para la tabla de logaritmo de seno se vuelve trivial, ya que la base se ha elegido cuidadosamente de tal forma que genere toda la gama de valores que abarca el rango dinámico de amplitud, incluidos aquellos valores entre 0.5 y 1 pasos de conteo (los cuales son redondeados hacia 1). Si bien esta tabla no genera el valor de 0, genera un valor mínimo, el cual podemos calcular de esta forma:

$$sen(\alpha_{min})=sen\left(\frac{0+0.5}{256}(2\pi)\right)$$
$$sen(\alpha_{min})=0.0122715383$$

Al normalizar este valor a una amplitud de 63, llegamos a la siguiente conclusión:

$$63(0.0122715383)=0.773106912 \ > \ 0.5$$

Como puede verse, el valor se encuentra por arriba de 0.5 pasos de conteo, lo cual implica que el logaritmo de este valor no excederá el límite superior de la tabla. Lo cual se comprueba al aplicar el logaritmo:

$$\log_b|sen\,\alpha_{min}|=\frac{\log(0.0122715383)}{\log(0.981212908)}=232.0213\approx 232$$

Este valor 232 se encuentra muy próximo al máximo de 255 y no lo excede, lo cual resulta idóneo puesto que se aprovecha muy bien el rango dinámico de entrada de la tabla exponencial. Si esto no hubiera sido así y el valor excediera el límite de entrada de la tabla exponencial la solución hubiera sido simple: se almacena en la tabla de logaritmo de seno el valor de 255 (se recorta) y se ajusta la base de la tabla exponencial para que su última posición contenga un cero por redondeo (aplicando una raíz con índice 254 en vez de 255).

Es importante señalar que no se debe ajustar el rango dinámico de esta tabla aplicando un factor de escalamiento (como por ejemplo 255/232). Esto causaría efectivamente un cambio en la base del logaritmo, por causa de la siguiente propiedad:

$$x \cdot \log_b y = \frac{\log_b y}{1/x}$$

$$x \cdot \log_b y = \frac{\log_b y}{\log_b b^{1/x}} \quad \leftarrow \quad \log_b b^x = x$$

$$x \cdot \log_b y = \frac{\log_b y}{\log_b \sqrt[x]{b}}$$

$$x \cdot \log_b y = \log_{\sqrt[x]{b}} y$$

Como puede verse, aplicar un factor de corrección “x” equivale a cambiar la base por la raíz de índice “x” de la base original. Esto causa distorsiones a la onda sinusoidal, introduciendo armónicos no deseados.

Por otra parte, los valores máximos de la función seno tienden a uno (aunque no exactamente por el factor 0.5 que se suma a n); es de esperar que al aplicar el logaritmo a estos valores se aproximen a cero, creando amplitudes máximas al aplicar la tabla exponencial. Esto ocurre de forma natural y sin necesidad de ajuste alguno.

Por las razones mencionadas con anterioridad, se concluye que la tabla de logaritmo de seno se construye de forma normalizada con tan sólo aplicar la base de logaritmo calculada para la tabla exponencial y sin aplicar factores de corrección. Quedando de la siguiente forma (también se deben redondear los valores obtenidos):

$$\log_b |sen\,\alpha| = \frac{\log \left| sen\left(\frac{0.5+n}{256} (2\pi) \right) \right|}{\log(0.981212908)} \quad , \quad 0 \le n \le 255$$

Finalmente, se invita al lector a que consulte la tabla adjunta (en formato de LibreOffice Calc) con todos los valores de las tablas así como una prueba del algoritmo donde se puede variar manualmente la amplitud de la onda generada. El programa completo se encuentra detallado y comentado en el sketch de Arduino, en el archivo de código fuente llamado sintesis-log-sen.ino.

Autor: Joksan Alvarado.
Repositorio git: www.github.com/joksan/sintetizador-arduino