Parte 3: Osciladores digitales básicos

En esta nueva sección comenzaremos a trabajar con los fundamentos de los osciladores para producir -eventualmente- nuestras señales tonales o de modulación, entre otras cosas. En Faust, como en muchos sistemas de audio digital, una señal tiene un rango de entre -1.0 y +1.0 (aunque esto no es estricto). Sin embargo, para nuestro primer ejemplo comenzaremos creando una señal tipo diente de sierra entre 0.0 y 1.0, que utilizaremos como ***generador de fase*** para producir diferentes formas de onda.

Contenidos | Parte 3

Parte 3: Osciladores digitales básicos 1

Generador de fase 1

Ejemplo 1: Rampa 2

Escritura matemática 3

Ejemplo 2: Una señal de fase 3

Controlando la frecuencia 4

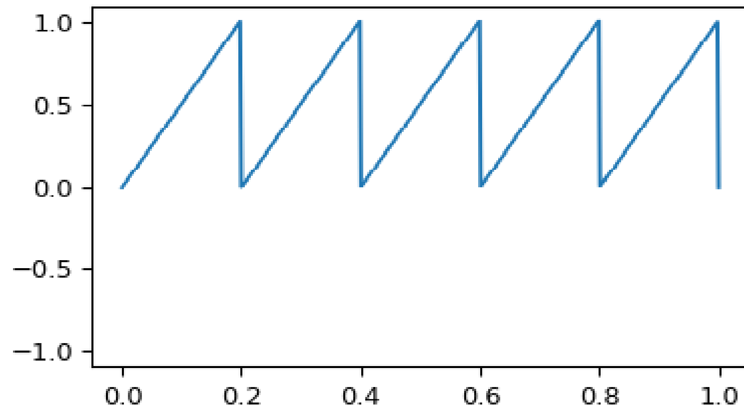
Ejemplo 3: Generador de señal de diente de sierra 5

Ejemplo 4: Generador de onda cuadrada 6

Generador de fase

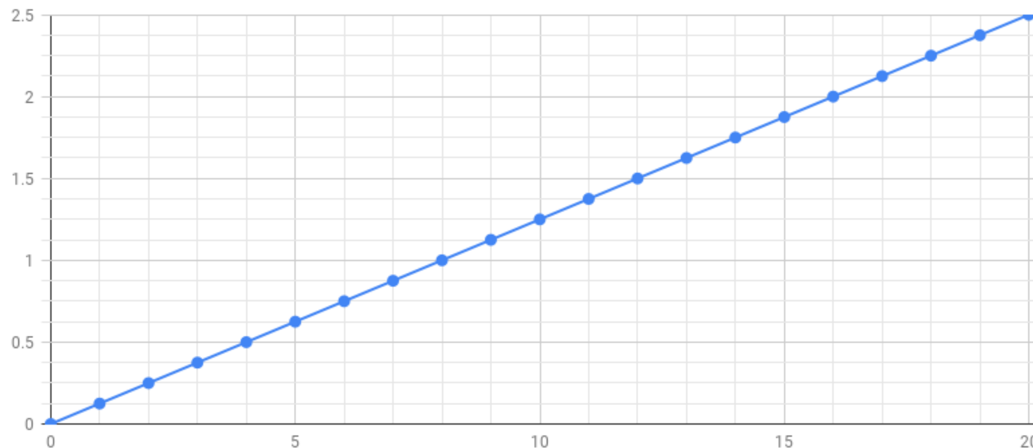
Si pensamos por un momento en funciones trigonométricas como el seno, coseno, etc, recordaremos que son funciones ***periódicas***, es decir, que se repiten tras un determinado lapso de tiempo (aunque puede ser otra variable), y de manera indefinida. También podremos recordar que generalmente definíamos un punto cualquiera de la función utilizando un ángulo, sin importar si lo expresábamos en radianes, grados sexagesimales, porcentajes, o hasta en manzanas… A este ángulo lo llamamos ***fase***, pero es simplemente una manera de expresar un punto (o "momento") del ciclo de la función periódica.

Un generador de fase nos permitirá precisamente generar una señal que crece de manera lineal, pero que al llegar a un máximo (que puede ser 1.0, o 360°, o 2π, etc…), se reiniciará a cero, de manera que podamos usarla para "recorrer" los valores de una función periódica o generar otras nuevas con algunas operaciones simples. En principio, el resultado del generador de fase es muy similar a una ***diente de sierra*** ideal:

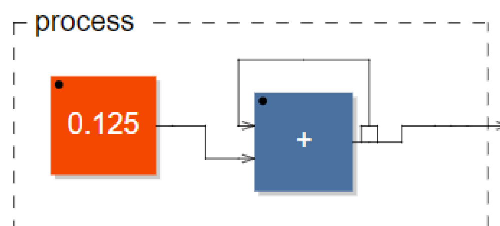


Ejemplo 1: Rampa

Para crear esta señal, comenzaremos produciendo una rampa "infinita", es decir, un número que crece indefinidamente. ¿Cómo? ¡Contando! Tal como contamos hasta el 10 o cualquier número, sumando un mismo intervalo de manera acumulativa: 0, 1, 2, 3… 10. Luego, para obtener nuestra "diente de sierra" transformaremos esta rampa infinita en una señal periódica gracias a una operación de ***parte decimal***, donde sólo retendremos los decimales de un número y descartaremos su parte entera.



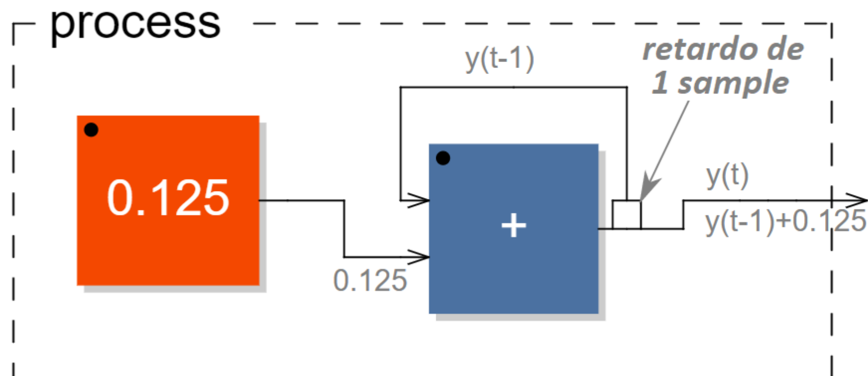
La rampa es producida por el siguiente programa. Observemos cómo aprovechamos la composición recursiva para sumar un número de manera acumulativa.




```
process = 0.125 : + ~ _;
```

Escritura matemática

Para entender mejor el diagrama anterior, explicaremos matemáticamente qué ocurre paso a paso.



Como se puede ver en el diagrama, la fórmula para la señal de salida es:

$$(t) = y(t - 1) + 0.125$$

Esta fórmula podemos describirla con estas palabras: “al resultado anterior, le sumamos 0.125”. Podemos calcular a mano los primeros valores de $y(t)$ y (t):

- $y(t < 0) = 0$
- $y(0) = y(-1) + 0.125 = 0.125$
- $y(1) = y(0) + 0.125 = 2 * 0.125 = 0.250$
- $y(2) = y(1) + 0.125 = 3 * 0.125 = 0.375$
- ...
- $y(6) = y(5) + 0.125 = 7 * 0.125 = 0.875$
- $y(7) = y(6) + 0.125 = 8 * 0.125 = 1.000$
- $y(8) = y(7) + 0.125 = 9 * 0.125 = 1.125$
- ...

Ejemplo 2: una señal de fase

¿Cómo convierto la rampa anterior en una señal de diente de sierra? Al descartar toda la parte entera de las muestras para mantener sólo la parte decimal: $3.14159 \rightarrow 0.14159$. Para ello podemos valernos de una función primitiva integrada en Faust (y en la mayoría de los lenguajes de programación), que convierte cualquier número en un entero: `int(x)`. La primitiva `int(x)` toma un número cualquiera, y lo convierte en un número entero sin decimales, *y aunque parece lo contrario a lo que precisamos*, combinándolo con una simple resta podemos usarlo para retener sólo la parte decimal. Crearemos una función para hacer esto:

```
// Al numero original con decimales, le restamos el mismo numero SIN decimales
// El resultado sera siempre la parte decimal del numero original
parteDecimal(x) = x - int(x);
```

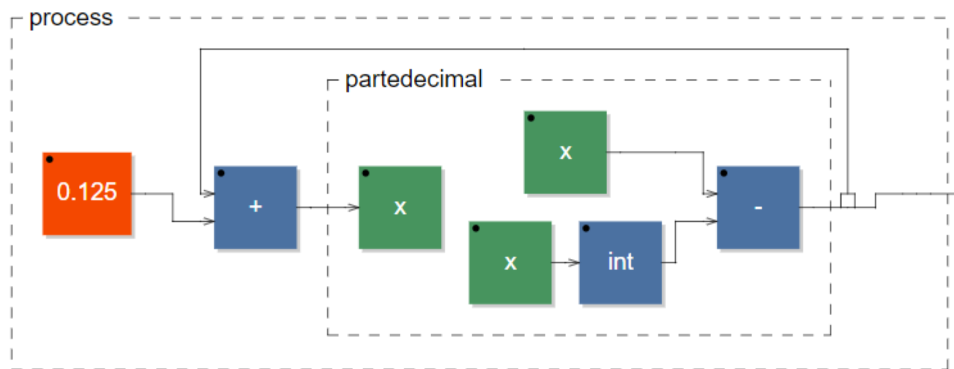
Ahora podemos usar esta función para convertir nuestra rampa en una ***diente de sierra***. Es tentador conectarla directamente con nuestra rampa infinita y escribir:

```
parteDecimal(x) = x - int(x);
process = 0.125 : + ~ _ : parteDecimal;   // cuidado con esto!
```

Desde un punto de vista matemático, eso sería perfectamente correcto, pero en la realidad de la informática digital… terminaremos acumulando pequeños errores de redondeo en cada conversión a enteros de `int(x)`. Para mantener la precisión total, es ***más conveniente*** colocar la operación de la parte decimal ***dentro del bucle***, así:

```
parteDecimal(x) = x - int(x);
process = 0.125 : (+ : parteDecimal) ~ _;
```

Ahora podemos probar todo el código (¡cuidado con el volumen!):




```
parteDecimal(x) = x - int(x);
fase = 0.125 : (+ : parteDecimal) ~ _;
process = fase;
```

Controlando la frecuencia

En nuestra definición de `fase`, el valor del paso que estábamos sumando (`0.125`) determina la ***frecuencia*** de la señal generada, pero evidentemente no se trata de un número en Hz que podamos controlar en forma directa. Para poder hacer eso, necesitamos calcular este valor de paso en función de la frecuencia que deseamos, y también de la ***frecuencia de muestreo*** del sistema, ya que ésta determinará la ***velocidad*** a la que funcionan los procesos de nuestro DSP.

Pero no basta con mirar la frecuencia de muestreo y tipearla en el código, sino que el programa deberá poder conocerla y adaptarse a cada sistema donde esté funcionando. Para ello contamos con la función `ma.SR`, pero no viene incluida por defecto: para utilizarla debemos incluir la ***librería estándar de Faust***, añadiendo la siguiente línea al comienzo del programa: `import("stdfaust.lib");`.

Supongamos que queremos que nuestra señal de fase tenga una frecuencia de 1 Hz, como para hacer las cuentas sencillas. Entonces el paso debe ser: `1/ma.SR` (frecuencia deseada sobre

frecuencia de muestreo), de modo que tome esa cantidad de muestras para que la señal de fase pase de 0.0 a 1.0. Si queremos una frecuencia de 440 Hz, necesitamos un paso 440 veces mayor para que la señal de fase pase de 0.0 a 1.0 unas 440 veces más rápido: `440/ma.SR`.

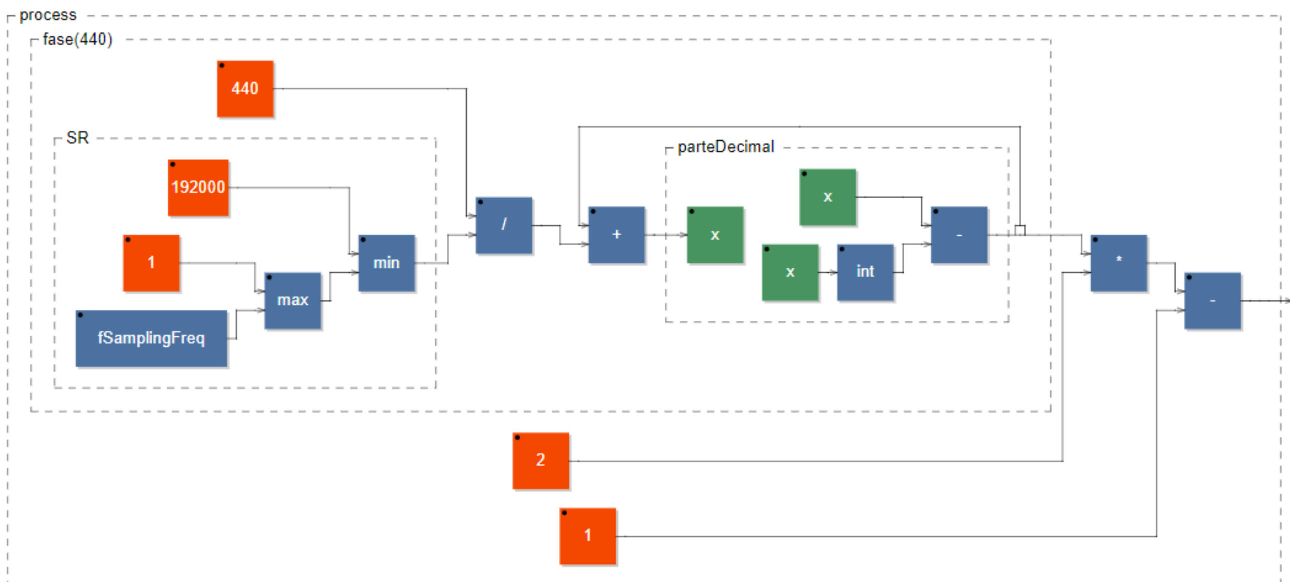
```
fase = 440/ma.SR : (+ : parteDecimal) ~ _;
```

Esta definición se puede generalizar reemplazando el número fijo `440` por un parámetro, `f`, de manera que podamos elegir directamente una frecuencia en Hz cuando utilicemos `fase(f)`.

```
import("stdfaust.lib");
parteDecimal(x) = x - int(x);
fase(f) = f/ma.SR : (+ : parteDecimal) ~ _;
process = fase(440);
```

Ejemplo 3: generador de señal de diente de sierra

Ahora podemos usar el generador de fase para producir distintas formas de onda, así que comencemos por una ***diente de sierra***. Si bien la señal de nuestro generador ya es idéntica a una diente de sierra, tiene un pequeño inconveniente: sus valores oscilan entre 0.0 y 1.0, en lugar de -1.0 y 1.0 como necesitamos en una señal de audio adecuada. Podemos resolver esto con dos simples cuentas: duplicamos la señal, para que oscile entre 0.0 y 2.0, y luego restamos 1, quedando finalmente entre -1.0 y 1.0 como deseábamos.

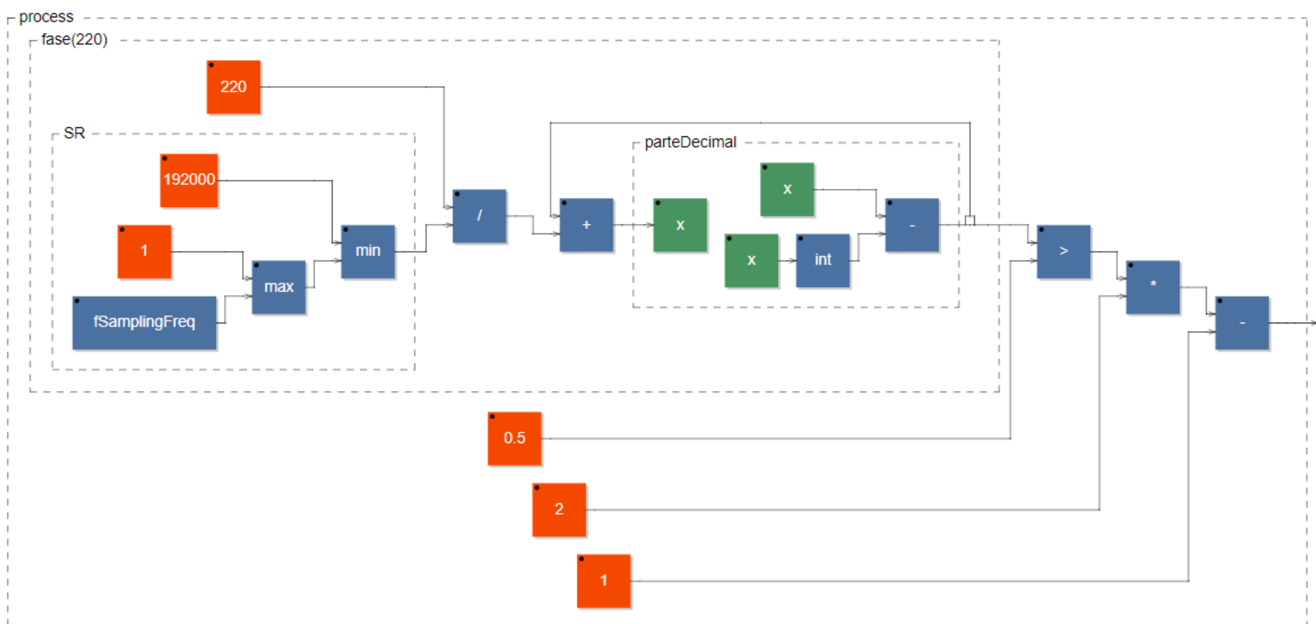


```
import("stdfaust.lib");

parteDecimal(x) = x - int(x);
fase(f) = f/ma.SR : (+ : parteDecimal) ~ _;

dienteSierra(f) = (fase(f) * 2) - 1;      // adaptamos el rango entre -1 y 1

process = dienteSierra(440);      // podemos probar otras frecuencias
```

Ejemplo 4: generador de onda cuadrada

También podemos usar el generador de fase para producir una señal de onda cuadrada. Nos valdremos para ello de la primitiva `>`, que compara si el número de la izquierda es mayor al de la derecha: `A > B`. `>` es una operación de ***comparación lógica***, lo que quiere decir que evaluará una expresión, y nos devolverá 1 si resulta verdadera, y 0 si resulta falsa. Podríamos interpretar `A > B` de la siguiente manera: "si A es mayor que B, entonces devuelve 1; de lo contario devuelve 0".

Solo para refrescar la memoria: una onda cuadrada oscila entre dos valores, ***uno máximo y uno mínimo***, pero sin pasar (en principio) por los valores intermedios, es decir, ***cambia de manera abrupta***. Teniendo nuestro generador de fase, que genera un ciclo de rampas entre 0.0 y 1.0, resultará sencillo convertirlo en una onda cuadrada utilizando la comparación lógica para evaluar todos sus valores y separarlos en dos. Sabemos que el punto medio entre 0.0 y 1.0 del generador de fase es 0.5, por lo que podemos hacer la siguiente comparación: si el valor es mayor al punto medio, ***que nos devuelva siempre 1***; y si es menor, ***que devuelva siempre 0***. De esta manera nuestra señal oscilará a la frecuencia del generador de fase, pero lo hará únicamente con los valores 0.0 y 1.0 sin los valores intermedios. Finalmente aplicamos la misma idea que con la señal diente de sierra para adaptar el valor al rango de -1.0 a 1.0.



```
import("stdfaust.lib");

parteDecimal(x) = x - int(x);
fase(f) = f/ma.SR : (+ : parteDecimal) ~ _;

ondaCuadrada(f) = ((fase(f) > 0.5) * 2) - 1;

process = ondaCuadrada(220);
```

Hemos visto cómo trabajar con osciladores digitales en su nivel más elemental. En la siguiente parte de este tutorial veremos cómo generar algunas formas de onda más y comenzaremos a experimentar realizando síntesis aditiva.