The Reaktor Core tutorial to accompany "Generation of bandlimited sync transitions for sine waveforms"

Vadim Zavalishin*

May 4, 2009

WARNING. This tutorial is not a beginner one. The knowledge of DSP theory as well as a good command of Reaktor Core is assumed.

Also, this is a tutorial and not a walkthrough. Be prepared to that a good amount of the material will be available only in the form of hints, with corresponding exercises.

1 Introduction

The article "Generation of bandlimited sync transitions for sine waveforms", which this tutorial is supposed to accompany, describes a set of DSP techniques to generate antialiased synced sine oscillators. The purpose of this tutorial is to provide examples of how the third of these techniques (frequency shifting) can be used within the Reaktor Core environment. The same implementation can be also used to build a (somewhat inefficient) implementation of the ring modulation technique.

2 Ins and outs

We are going to create a Core cell generating a synced sine. We will also include the master oscillator inside the Core cell, although it should represent no difficulty using an external master signal instead.

Therefore we will need an input for the master oscillator frequency in Hz, an input for the slave oscillator frequency in Hz, an input for the slave oscillator reset phase ($0 \ldots 1$), and the slave oscillator's output. We will also output the master oscillator's phase signal, which can be used as an external sync signal for an oscilloscope:

3 Master oscillator

The master oscillator needs to produce the transition-triggering events, where the value of the events contains the subsample position (0 to 1) of the transition time moment. This can be achieved by combining the *Phase* macro, which can be found e.g. inside the *Sin Osc* standard macro, with the *Sync* macro found inside the *4-Wave Slv* standard macro:

4 Quad oscillator

To generate the analytic difference of two sine waveforms we need a quad oscillator, producing phase-locked sine and cosine signals. Also, by multiplying the complex output of such oscillator with a complex amplitude, a sine of an arbitrary phase can be

*Native Instruments GmbH

generated:

$$\cos(\omega t + \varphi) = \mathrm{Re}\left(e^{j\varphi} e^{j\omega t}\right)$$

In order to avoid the algorithm quality issues related to the quality of the oscillator itself (and particularly in order to be able to use the above formula without much thinking) we are going to employ a sine approximation with a very low SNR (-112dB). First we build a polynomial approximation of $\sin \pi x$ on $[0, 0.5]$ by the 3rd-order Hermite interpolation. Moving out the factor x and taking the absolute value we get an approximation working on $[-0.5, 0.5]$:

$$\sin \pi x \approx p_7(x)$$

where

$$\begin{aligned} p_7(x) = x \cdot \Big| & -0.4094245 \cdot |x|^6 - 0.1867857 \cdot |x|^5 \\ & + 2.61945 \cdot |x|^4 - 0.009166345 \cdot |x|^3 \\ & - 5.167713 \cdot |x|^2 + 3.141593 \Big| \end{aligned}$$

Then

$$\cos 2\pi x \approx p_7\left(0.5 - |2x|\right) \qquad (-0.5 \le x \le 0.5)$$

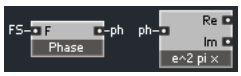
and we use the equality

$$\sin 2\pi x = \cos 2\pi\big((x - 0.25) - \lfloor x - 0.25 \rceil\big)$$

(where $\lfloor\,\rceil$ denotes rounding) to compute the sine counterpart.

We combine both formulas to build a macro producing the real and imaginary parts of $e^{2\pi j x}$. Note that the above formula for the cosine works only on $[-0.5, 0.5]$, so we need an additional wrapper at the input.

The quad oscillator signal can then be built by using the $e^{2\pi j x}$ module and the *Phase* macro found e.g. inside the *Sin Osc* standard macro:



The phase signal will also be required to generate the new phase for the sync transition.

5 Transition info

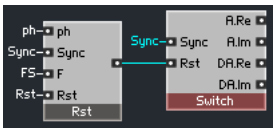
For simplicity, we will assume that the sync transitions do not overlap. We are going to use 20 samples-long transitions, therefore the master oscillator frequency is limited to 1/20 of the sampling rate, which at 44kHz is approximately 3 octaves above the middle C. Then only one aliasing residual and only one analytic difference ΔX need to be generated at a given time moment. Also only two sine waveforms with different phases (the "old" signal $X_1(t)$ and the "new" signal $X_2(t)$) are simultaneously involved:

$$\begin{aligned} X(t) &= e^{j\omega t} \\ X_1(t) &= e^{j\varphi_1} X(t) = A_1 X(t) \\ X_2(t) &= e^{j\varphi_2} X(t) = A_2 X(t) \\ \Delta X(t) &= (A_2 - A_1)\, X(t) = \Delta A\, X(t) \end{aligned}$$

where $X(t)$ is the "base" signal, used for the generation of both $y(t)$ and $\Delta X(t)$. Then

$$\begin{aligned} A(t) &= \begin{cases} A_1 & \text{if } t < 0 \\ A_2 & \text{if } t \ge 0 \end{cases} \\ y(t) &= \mathrm{Re}\big(A(t)X(t)\big) \\ \bar{y}(t) &= y(t) - \mathrm{Re}\big(\Delta\bar{h}(t)\Delta A\, X(t)\big) \end{aligned}$$

The switching of the value of $A(t)$ can be done simply in response to the transition triggering signal. Simultaneously a new value of ΔA can be generated. So we build a macro called *Switch*, which does exactly this job:



Upon receiving an event at the *Sync* input, the macro replaces the old value of φ_1 with the old value of φ_2 and sets φ_2 to the value at the *Rst* input. The corresponding real and imaginary parts of $A = A_2$ and ΔA are sent to the outputs.

The value at the *Rst* input of the *Switch* macro is basically equal to the difference of the value at the *Rst* input of the entire Core cell and the current

slave oscillator phase *ph*. However, one also needs to take into account the amount of time "passed" since the transition, which is equal to the value of the *Sync* event multiplied by the slave oscillator frequency and divided by sampling rate. This is exactly what the *Rst* macro is doing.

6 Table lookup

To generate $\bar{h}(t)$ we need two tabulated functions: the window function and the Ein function.[1] We tabulate the Kaiser window function

$$W(x) = \begin{cases} \dfrac{I_0(\alpha\sqrt{1-x^2})}{I_0(\alpha)} & \text{if } |x| \le 1 \\ 0 & \text{otherwise} \end{cases}$$

(with $\alpha = 4$) and its derivative at $x = 0,\ 0.1,\ 0.2,\ \ldots,\ 1$, taking 11 samples in total. Using a 1st-order Hermite interpolation, we have SNR under -100dB.

We also create the table of $(\mathrm{Ein}\,\pi jx)/(2\pi j)$ function and its derivative on $[0, 2]$. However here more samples are required: $x = 0,\ 0.02,\ 0.04,\ \ldots,\ 2$, providing SNR of about -95dB. Since the function is complex, we sample its real and imaginary parts into two separate tables.

In each of the three tables, we interleave the function and derivative values. Since all tables have the same structure, it makes sense to use the same macro for the table lookup. There is a difference however, in that the window function and the imaginary part of $(\mathrm{Ein}\,\pi jx)/(2\pi j)$ are even, while the real part of $(\mathrm{Ein}\,\pi jx)/(2\pi j)$ is odd. So, for $x < 0$ we need a slightly different handling and thus we need two different macros, one for the "odd" interpolation and one for the "even" one.

The contents of the *Odd* macro are shown in Fig. 1. The *Pos* macro generates the lookup position. Upon receiving an event at its *Rst* input, the *Pos* macro initializes the position to the received value, and then increments the position by *Inc* on each subsequent *SR.C* clock (note that *Inc* may be negative!). The position never gets incremented past ±Max due to internal clipping.

[1] The article text contains references to the texts explaining how to compute the Ein function, as well as the modified Bessel functions, which are needed to compute the Kaiser window.

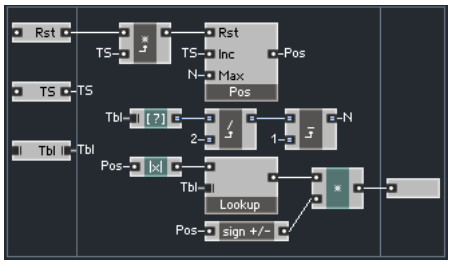


Figure 1: The *Odd* macro.

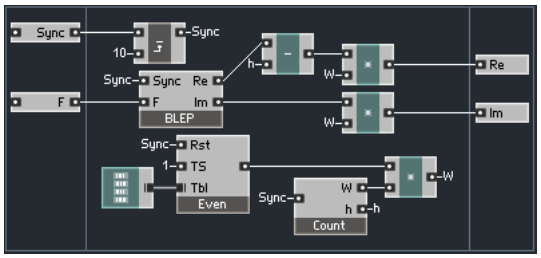


Figure 2: The internal structure of the *Residual* macro.

The input *TS* provides the time-scaling value, which scales the *Rst* input as well as provides the increment value. The *Lookup* macro does the interpolation in the table, according to the position (note that the *Lookup* macro still needs to do some internal position clipping!). The sign inversion part ensures that the function is odd (obviously the corresponding modules are not there in the *Even* macro).

7 Aliasing residual

Now we need to generate the residual signal $\Delta\bar{h}(t)$. Instead, we will generate the inverted signal

$$-\Delta\bar{h}(t) = \bar{h}(t) - h(t)$$

so that

$$\bar{y}(t) = y(t) + \mathrm{Re}\left(-\Delta\bar{h}(t)\Delta A\, X(t)\right)$$

The generation of the signal $-\Delta\bar{h}(t)$ is done by the *Residual* macro whose internal structure is shown in

Fig. 2. First, the value of the event at the *Sync* input event is converted into a real sample offset from the transition time by subtracting 10 from the value. This value is then used to restart the *Even* macro, which reads out the window table. The BLEP macro (using internally *Odd* and *Even* macros) does the same for the $(\mathrm{Ein}\,\pi jx)/(2\pi j)$ function, plus it computes the logarithm of the frequency ratio (in response to the *Sync* event), thereby producing the $\bar{h}(t)$ signal. Note that the *TS* coefficient for the $(\mathrm{Ein}\,\pi jx)/(2\pi j)$ lookup is equal to 5.

The *Count* macro has an internal integer counter, which is initialized to -10 upon receiving an input event. Simultaneously, the outputs are initialized to $W = 1$ and $h = -0.5$. The counter is then incremented on each subsequent *SR.C* clock. When the counter reaches 0, the h output is set to 0.5. When the counter reaches 10, the W output is set to 0 and the incrementing is stopped. Therefore the h output produces the signal $h(t)$, while the W output is used to switch the window signal to zero.

8 Final touches

Now we need to combine all modules together into a single structure. In doing so, recall that the signal $-\Delta\bar{h}(t)$ is not causal, and, unless we do something about it, it is coming 10 samples late. Therefore we need to introduce a 10 samples delay on the remaining signals $A(t)$ and $X(t)$ (the easiest way to build such delay is to modify the *Delay 1p* standard macro). We can also do a small optimization, where instead of delaying the complex signal A we can delay the real signal $y(t) = \mathrm{Re}(AX)$, thereby sparing one delay macro.

9 Homework

Questions and exercises

1. Implement all macros mentioned in the text and combine them into the Core cell. Output the result to the speakers and to the oscilloscope.
2. Make a minor modification to the structure, turning it into a non-antialiased synced sine. Compare the difference in the sound.
3. Introduce the parameter clipping into the Core cell, ensuring that the frequencies do not get out of the allowed range.
4. Vary the band limit of the implementation.
5. Modify the implementation so that it uses an external master signal.
6. Modify the implementation to allow 2 (or more, if you wish so) overlapping transitions.
7. Make a minor change to the implementation, which transforms it (inefficiently) into a ring modulation method implementation.
8. Make an efficient implementation of the ring modulation method.
9. Implement the multiple BLEPs method.

Acknowledgements

REAKTOR® and *REAKTOR Core Technology*® are registered trademarks of Native Instruments GmbH.