ALIAS-FREE OSCILLATOR SYNCHRONIZATION VIA ADDITIVE SYNTHESIS

Jonas Roth, Domenic Keller, Oscar Castañeda, and Christoph Studer

Department of Information Technology and Electrical Engineering, ETH Zurich, Switzerland
joroth@ethz.ch | domkeller@ethz.ch | caoscar@ethz.ch | studer@ethz.ch

ABSTRACT

Oscillator synchronization is a widely used sound-synthesis technique, but straightforward digital implementations suffer from aliasing artifacts. This paper presents an alias-free method for digital emulation of oscillator synchronization of arbitrary periodic waveforms based on additive synthesis. Starting from a finite set of Fourier-series coefficients representing a bandlimited free-running waveform, we derive linear spectral-resampling transforms that map these coefficients to those of the bandlimited synchronized waveform. Beyond conventional hard synchronization, the proposed approach also supports two additional soft-synchronization modes. To address the high computational complexity of the proposed method, we introduce HASY, a 6 mm^2 application-specific integrated circuit (ASIC) fabricated in 65 nm CMOS technology. HASY generates one 96 kHz, 24 bit alias-free synchronized waveform with up to 512 harmonics and computes the spectral-resampling transform within only five audio-sample periods.

1. INTRODUCTION

Oscillator synchronization is a widely used technique in music synthesizers that is able to create rich, expressive waveforms with intricate harmonic content. Examples of tracks prominently featuring oscillator synchronization are "Release the Beast" by Breakwater [1] and "Fletch Theme" by Harold Faltermeyer [2]. The synthesis method involves two oscillators, a *leading* oscillator and a *following* oscillator, which have fundamental frequencies f_{lead} and f_{follow}, respectively.[1] The classic oscillator-synchronization setup, called hard synchronization ("hard sync" for short), is illustrated in Figure 1. Whenever the leading oscillator completes one period (i.e., on the rising edge of the waveform), then the following oscillator's phase is reset (i.e., the oscillator starts its waveform anew). Depending on the following oscillator's amplitude at the reset instant, this reset procedure will introduce a discontinuity in the waveform, causing new high-frequency components that were not present in the free-running following-oscillator waveform.

Remark 1. *Conceptually, the leading oscillator's waveform is unimportant. However, in a traditional analog synthesizer, phase resets take place on rising edges of the leading oscillator. Thus, we depict the leading oscillator in Figure 1 with a sawtooth wave.*

Alternative modes to hard sync also exist. Soft oscillator synchronization ("soft sync" for short) is a generic term for variants

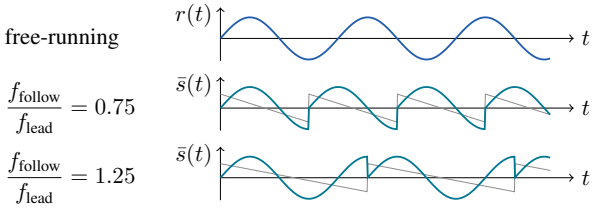


Figure 1: Illustration for hard sync with a sine wave for the following-oscillator waveform. Top: free-running following oscillator. Middle and bottom: (petrol) hard-sync signal and (gray) exemplary leading oscillator. The middle and bottom each use a different following-to-leading-oscillator frequency ratio.

that modify the following oscillator's reset procedure in order to control the time-domain discontinuity induced by synchronization. In practice, soft-sync implementations replace the hard phase reset with rules such as waveform reversal at the reset instant (often referred to as *reversing sync* or *mirrored sync*), or with conditional reset procedures. These modifications affect the spectral composition of the synchronized waveform and often help reduce the perceived "harshness" associated with hard sync.

1.1. Digital Implementation of Hard Sync Is Hard

While analog circuit implementation of oscillator synchronization is rather straightforward, trivial digital emulations, e.g., with a hard phase reset, cause severe aliasing artifacts, as pointed out in [3]. The underlying issue is not specific to hard sync: any waveform with discontinuities is not bandlimited, and its high-frequency content aliases when sampled naïvely. A brute-force remedy that combines extreme oversampling with a sharp anti-aliasing low-pass filter incurs high computational overhead. Below, we discuss more efficient approaches that synthesize a bandlimited signal directly; see [4] for a general overview of anti-aliasing synthesis.

Early work on anti-aliasing synthesis focused on general-purpose techniques that reduce discontinuity-induced aliasing: BLIT [5] targets classical analog waveforms (sawtooth, square, and triangle); BLEP [3] was introduced to address hard sync. A later refinement of BLIT [6] adds a perceptual analysis of aliasing and applies the method to oscillator synchronization. More specialized methods have been proposed for particular oscillator types: For example, [7] applies anti-aliasing FIR low-pass filters to a synchronized sine wave, while [8] derives an efficient implementation for synchronized sawtooth oscillators from an analytical Fourier-series description. However, these approaches rely on waveform-specific derivations, which lack the flexibility of synchronizing *arbitrary* periodic waveforms. More general anti-aliasing methods, such as polynomial transition regions [9], have also been used to smooth the discontinuities at each reset.

[1] In historic terminology, the leading and following oscillators were called "master" and "slave" oscillators, respectively.

A more general approach to anti-aliasing is additive synthesis, which enables bandlimited synthesis of arbitrary periodic waveforms. However, an oscillator-based implementation is often considered computationally too expensive for software realizations. Less computationally intensive approaches for bandlimited synthesis instead rely on wavetables (e.g., pre-filtered waveforms at different fundamental frequencies) or methods such as inverse fast Fourier transform (IFFT) synthesis [10, pp. 458–467], [11]. Alternatively, oscillator-based additive synthesis can be implemented in hardware through application-specific integrated circuits (ASICs), as demonstrated in [12, 13]. However, as pointed out in [14], a true emulation of oscillator synchronization via additive synthesis would benefit from a spectral transformation describing the synchronized waveform for arbitrary inputs.

1.2. Contributions

We propose a general method to digitally emulate alias-free oscillator synchronization for arbitrary periodic free-running following-oscillator waveforms. In particular, given the Fourier-series coefficients of a bandlimited following-oscillator waveform, we compute the Fourier-series coefficients of the synchronized waveform via a spectral-resampling transform. We then generate an alias-free time-domain synchronized signal through additive synthesis. Our approach naturally supports three different oscillator-synchronization modes: (i) traditional hard sync, (ii) mirrored sync, and (iii) pulsar sync (related to pulsar synthesis [15]). Since the proposed spectral-resampling transform exhibits high complexity, we present HASY (short for hard sync), a real-time digital implementation on an ASIC. HASY implements spectral resampling and additive synthesis in real time for one following oscillator with up to 512 harmonics at a sample rate of 96 kHz with 24 bit resolution.

2. SPECTRAL RESAMPLING AND ADDITIVE RESYNTHESIS FOR OSCILLATOR SYNCHRONIZATION

Our approach for alias-free oscillator synchronization operates in the Fourier-series domain. Given a finite set of Fourier-series coefficients $\{c_n\}_{n=-N}^{N}$ describing a bandlimited free-running periodic following-oscillator waveform, we compute the corresponding Fourier-series coefficients $\{\bar{c}_n\}_{n=-N}^{N}$ of the synchronized waveform with leading-oscillator period $T_{\text{lead}} = 1/f_{\text{lead}}$. We call this linear mapping *spectral resampling*, which forms the core idea of our approach. With the transformed Fourier-series coefficients, we then generate the synchronized time-domain signal via additive synthesis — here, we restrict the number of synthesized harmonics as a function of T_{lead} and the sample rate f_{s} to avoid aliasing.

Remark 2. *Our approach computes the periodic waveform resulting from oscillator synchronization and does* not *implement a phase-synchronous reset in the time domain.*

Remark 3. *If $T_{\text{lead}} = 1/f_{\text{lead}}$ is modulated rapidly over time, then our approach may still induce aliasing artifacts each time the transform is recomputed.*

We now detail our method for hard sync and then outline the other two synchronization modes: mirrored sync and pulsar sync.

2.1. Spectral Resampling for Hard Sync

Let $\{c_n\}_{n=-N}^{N}$ be a set of complex-valued Fourier-series coefficients that describe the output signal from a free-running (i.e., unsynchronized) following oscillator with period T_{follow}:

$$r(t) = \sum_{n=-N}^{N} c_n \mathrm{e}^{\mathrm{j}\frac{2\pi}{T_{\text{follow}}}nt}, \quad t \in \mathbb{R}. \tag{1}$$

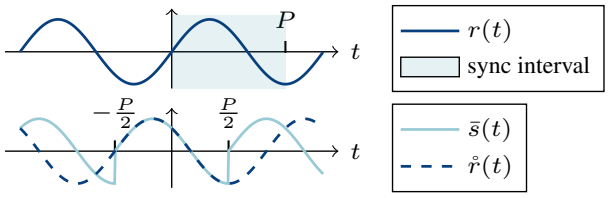


Figure 2: Illustration of the time shift operation with a sine wave as following-oscillator waveform and $P = 0.75$. Top: free-running following-oscillator waveform $r(t)$ with the desired sync interval marked. Bottom: time-shifted following-oscillator signal $\mathring{r}(t)$ and resulting hard-sync signal $\bar{s}(t)$ for alignment with the symmetrical Fourier integration window.

Remark 4. *Since N in (1) is finite, the free-running following-oscillator signal $r(t), t \in \mathbb{R}$ is bandlimited to N harmonics. This also implies that our approach will emulate oscillator synchronization applied to an already bandlimited following-oscillator waveform. Note that we use $r(\cdot)$ to denote $r(t), t \in \mathbb{R}$.*

Remark 5. *We rely on the complex-coefficient formulation to simplify our derivations. Since the produced time-domain signals $r(\cdot)$ are assumed to be real-valued, their Fourier-series coefficients $\{c_n\}_{n=-N}^{N}$ exhibit complex-conjugate symmetry $c_{-n} = c_n^*$. At the end of our calculations, we convert the complex-valued Fourier-series coefficients to the real-valued Fourier-series coefficients $\{a_0, \{a_n, b_n\}_{n=1}^{N}\}$ through the relationships $a_0 = c_0$, $a_n = 2\Re\{c_n\}$ and $b_n = -2\Im\{c_n\}$, where $\Re\{c_n\}$ and $\Im\{c_n\}$ are the real and imaginary parts of c_n, respectively.*

We now aim to emulate hard sync with a leading-oscillator period T_{lead}. In the Fourier-series coefficient domain, only the ratio of f_{follow} to f_{lead} matters. Therefore, we introduce the period ratio

$$P \triangleq \frac{T_{\text{lead}}}{T_{\text{follow}}} = \frac{f_{\text{follow}}}{f_{\text{lead}}}. \tag{2}$$

Without loss of generality, we can normalize the periods to

$$T_{\text{follow}} = 1 \quad \text{and} \quad T_{\text{lead}} = P, \tag{3}$$

which will simplify our derivations. With this normalization, we divide the Fourier-series coefficient transform into two parts: (i) a pre-rotation followed by (ii) the spectral-resampling transform.

2.1.1. Pre-Rotation

When illustrating oscillator synchronization, one usually considers the leading-oscillator period $[0, T_{\text{lead}}]$, which becomes $[0, P]$ after the normalization in (3). However, the spectral-resampling calculations simplify if we consider the interval $[-P/2, P/2]$ as the leading-oscillator period. To emulate oscillator synchronization correctly, the start of the following-oscillator waveform should align with the start of the leading-oscillator period (see Figure 2, top). To this end, we time-shift the following-oscillator signal as $\mathring{r}(t) \triangleq r(t - \tau)$ with $\tau = -P/2$ (see Figure 2, bottom). In the Fourier-series coefficient domain, this time shift corresponds to a

phase shift, and the "pre-rotated" coefficients $\{\mathring{c}_n\}_{n=-N}^{N}$ associated with the time-shifted signal $\mathring{r}(\cdot)$ are given by

$$\mathring{c}_n = c_n \mathrm{e}^{-\mathrm{j}2\pi\tau n/T_{\text{follow}}} = c_n \mathrm{e}^{-\mathrm{j}2\pi\tau n}, \quad n = -N, \ldots, N. \quad (4)$$

Note that we can apply the phase shift from (4) to the real and imaginary parts of the Fourier-series coefficients as

$$\begin{bmatrix} \Re\{\mathring{c}_n\} \\ \Im\{\mathring{c}_n\} \end{bmatrix} = \begin{bmatrix} \cos(2\pi n\tau) & \sin(2\pi n\tau) \\ -\sin(2\pi n\tau) & \cos(2\pi n\tau) \end{bmatrix} \begin{bmatrix} \Re\{c_n\} \\ \Im\{c_n\} \end{bmatrix}. \quad (5)$$

2.1.2. Spectral Resampling

We are now ready to calculate the complex-valued Fourier-series coefficients $\{\bar{c}_n\}_{n=-N}^{N}$ of the same signal $\mathring{r}(\cdot)$ but over a new period $T_{\text{lead}} = P$ instead of $T_{\text{follow}} = 1$. This operation is equivalent to synchronizing (resetting) the following oscillator at a rate of $1/P$, which will produce the desired hard-sync oscillator signal $\bar{s}(\cdot)$. The nth complex-valued Fourier-series coefficient associated with the synchronized waveform $\bar{s}(\cdot)$ is calculated as

$$\bar{c}_n = \frac{1}{P}\int_{-P/2}^{P/2} \mathring{r}(t)\mathrm{e}^{-\mathrm{j}\frac{2\pi}{P}nt}\mathrm{d}t \quad (6)$$

$$= \frac{1}{P}\int_{-P/2}^{P/2} \sum_{k=-N}^{N} \mathring{c}_k \mathrm{e}^{\mathrm{j}2\pi kt}\mathrm{e}^{-\mathrm{j}\frac{2\pi}{P}nt}\mathrm{d}t \quad (7)$$

$$= \sum_{k=-N}^{N} \mathring{c}_k \frac{1}{P}\int_{-P/2}^{P/2} \mathrm{e}^{\mathrm{j}2\pi(k-\frac{n}{P})t}\mathrm{d}t \quad (8)$$

$$= \sum_{k=-N}^{N} \mathring{c}_k \,\mathrm{sinc}(n - kP), \quad (9)$$

where we define the sinus cardinalis (sinc) function as

$$\mathrm{sinc}(x) \triangleq \begin{cases} \frac{\sin(\pi x)}{\pi x} & \text{if } x \neq 0 \\ 1 & \text{if } x = 0. \end{cases} \quad (10)$$

In essence, the spectral-resampling transform in (9) takes in all $2N+1$ Fourier-series coefficients $\{\mathring{c}_k\}_{k=-N}^{N}$ and produces a new set of Fourier-series coefficients $\{\bar{c}_n\}_{n=-N}^{N}$, which describe the hard-sync waveform with the leading-to-following-oscillator period ratio P. Mathematically, (9) corresponds to a *linear* transform that maps the Fourier-series coefficients of the pre-rotated (i.e., time-shifted) following-oscillator waveform $\{\mathring{c}_n\}_{n=-N}^{N}$ to the coefficients of the hard-synchronized waveform $\{\bar{c}_n\}_{n=-N}^{N}$.

Remark 6. *In (9), we map $2N+1$ input to $2N+1$ output coefficients. Alternatively, one could compute an output coefficient set of different size by evaluating (9) for $n = -N', \ldots, N'$. This would enable individual control of the input and output bandlimiting.*

In audio applications, one is typically interested in synthesizing a real-valued time-domain signal. To this end, we convert the complex-valued Fourier-series coefficients $\{\bar{c}_n\}_{n=-N}^{N}$ into the corresponding real-valued Fourier-series coefficients $\{\bar{a}_0, \{\bar{a}_n, \bar{b}_n\}_{n=1}^{N}\}$, removing the complex conjugate redundancy of real-valued signals. Instead of first computing the complex-valued coefficients and then transforming them into the real domain, we can directly compute the linear spectral-resampling transform on the real-valued coefficients for $n = 1, \ldots, N$:

$$\bar{a}_0 = \mathring{a}_0 + \sum_{k=1}^{N} \mathring{a}_k \,\mathrm{sinc}(kP) \quad (11)$$

$$\bar{a}_n = \sum_{k=1}^{N} \mathring{a}_k \big(\mathrm{sinc}(n - kP) + \mathrm{sinc}(n + kP)\big) \quad (12)$$

$$\bar{b}_n = \sum_{k=1}^{N} \mathring{b}_k \big(\mathrm{sinc}(n - kP) - \mathrm{sinc}(n + kP)\big). \quad (13)$$

Note that the pre-rotation (time shift) must first be carried out using the explicit method shown in (5), which can also be applied to the real-valued Fourier-series coefficients.

2.2. Additive Resynthesis

With the real-valued Fourier-series coefficients of the hard-sync output signal, we synthesize an alias-free time-domain signal using the approach from [13]. Concretely, the alias-free discrete-time hard-sync output signal $\bar{s}[\cdot]$ at sample rate f_s is given by

$$\bar{s}\,[\ell] = \bar{a}_0 + \sum_{n\in\mathcal{N}} \bar{a}_n \cos\!\left(\frac{2\pi n}{T_{\text{lead}} f_\text{s}}\ell\right) + \bar{b}_n \sin\!\left(\frac{2\pi n}{T_{\text{lead}} f_\text{s}}\ell\right), \quad (14)$$

where we only consider harmonics in the set

$$\mathcal{N} \triangleq \{n = 1, \ldots, N : n/T_{\text{lead}} < f_\text{s}/2\}. \quad (15)$$

In other words, we only sum harmonics below the Nyquist frequency $f_\text{s}/2$. This is why our approach is alias-free.

Remark 7. *In (14), we reintroduced the period T_{lead} of the leading oscillator, abandoning the normalization in (3) and setting T_{lead} to arrive at the desired leading-oscillator fundamental frequency.*

Remark 8. *In contrast to the additive-synthesis approach in [13], we do not allow arbitrary partials. Instead, we restrict ourselves to pure harmonics, i.e., integer multiples of the fundamental.*

Remark 9. *Due to the pre-rotation (time shift) from Section 2.1.1, the reset instant in the hard-sync output signal takes place at $t = -T_{lead}/2$, i.e., the output signal appears time-shifted by $\tau = -P/2$. If such a time shift is undesired, one could either apply a post-rotation to the coefficients $\{\bar{c}_n\}$ or undo the time shift during additive synthesis (14). In our implementation, we ignore such a post-processing step entirely.*

2.3. Spectral Resampling for Mirrored Sync

Our resampling method naturally extends to other oscillator-synchronization methods, such as *mirrored sync*. Instead of resetting the following oscillator as for hard sync, mirrored sync "reflects" the following-oscillator waveform at the reset instant, as illustrated in Figure 3. Mathematically, the mirrored-sync output signal $\hat{s}(\cdot)$ has a total period of $2\,T_{\text{lead}}$ (T_{lead} for the forward and T_{lead} for the mirrored waveform) and can be described as

$$\hat{s}(t) = \begin{cases} r(t) & \text{if } 0 \le t < T_{\text{lead}} \\ r(2T_{\text{lead}} - t) & \text{if } T_{\text{lead}} \le t < 2\,T_{\text{lead}}. \end{cases} \quad (16)$$

Following the same approach as in Section 2.1 for hard sync, we can derive the complex-valued Fourier-series coefficient transform

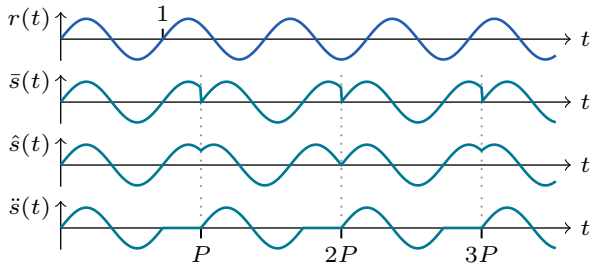


Figure 3: Comparison of different sync modes, here using a sinusoidal following-oscillator waveform and $P = \frac{11}{8}$. From top to bottom: (free-running) following oscillator $r(t)$, hard sync $\bar{s}(t)$, mirrored sync $\hat{s}(t)$, and pulsar sync $\ddot{s}(t)$.

for the mirrored sync. Again, we assume $T_{\text{follow}} = 1$ and $T_{\text{lead}} = P$. Starting from the real-valued coefficients $\{a_0, \{a_n, b_n\}_{n=1}^N\}$, we first apply the pre-rotation in (5) with $\tau = -P$ to get $\{\mathring{a}_0, \{\mathring{a}_n, \mathring{b}_n\}_{n=1}^N\}$. We then derive the spectral-resampling transform for the mirrored sync analogous to Section 2.1.2, yielding the coefficients $\{\hat{a}_0, \{\hat{a}_n, \hat{b}_n\}_{n=1}^N\}$ that describe the mirrored-sync signal $\hat{s}(\cdot)$. The resulting transform for real-valued Fourier-series coefficients is

$$\hat{a}_0 = \mathring{a}_0 + \sum_{k=1}^{N} \mathring{a}_k \,\mathrm{sinc}(2kP) - \mathring{b}_k \mathrm{versinc}(2kP) \tag{17}$$

$$\hat{a}_n = \sum_{k=1}^{N} \Big[\mathring{a}_k \big(\mathrm{sinc}(n-2kP) + \mathrm{sinc}(n+2kP)\big) + \mathring{b}_k \big(\mathrm{versinc}(n-2kP) - \mathrm{versinc}(n+2kP)\big) \Big] \tag{18}$$

$$\hat{b}_n = 0, \tag{19}$$

for $n = 1, \ldots, N$. Here, we define the *versed sinus cardinalis* (versinc) function[2] as follows:

$$\mathrm{versinc}(x) \triangleq \begin{cases} \frac{1-\cos(\pi x)}{\pi x} & \text{if } x \neq 0 \\ 0 & \text{if } x = 0. \end{cases} \tag{20}$$

Remark 10. *In contrast to (11)–(13) for hard sync, the resulting linear transform in (17), (18) also features* versinc$(\cdot)$ *terms. Nonetheless, the implementation similarity between the sinc and versinc functions enables, as discussed in Section 3, a shared hardware implementation of both hard and mirrored sync.*

Remark 11. *The Fourier-series coefficients* $\{\hat{b}_n\}_{n=1}^N$ *in (19) vanish, which is a direct consequence of the even symmetry in the mirrored-sync signal* $\hat{s}(\cdot)$.

In order to arrive at an alias-free output signal $\hat{s}[\ell]$ for the mirrored sync, we use the same additive-synthesis approach from Section 2.2 with the Fourier-series coefficients $\{\hat{a}_0, \{\hat{a}_n, \hat{b}_n\}_{n=1}^N\}$. However, since $\hat{s}(\cdot)$ has a period of $2\,T_{\text{lead}}$, one must utilize $2\,T_{\text{lead}}$ instead of T_{lead} in Section 2.2.

2.4. Spectral Resampling for Pulsar Sync

Another alternative that fits naturally in our spectral-resampling framework is what we call *pulsar sync*. Instead of resetting the following oscillator as hard sync does, pulsar sync periodically mutes the following oscillator after each following-oscillator period T_{follow}. Therefore, this mode only makes sense for $P \geq 1$. Mathematically, one period of the pulsar-sync output signal $\ddot{s}(t)$ with period T_{lead} can be described as

$$\ddot{s}(t) = \begin{cases} r(t) & \text{if } 0 \leq t < T_{\text{follow}} \\ 0 & \text{if } T_{\text{follow}} \leq t < T_{\text{lead}}. \end{cases} \tag{21}$$

Figure 3 features an illustration of the resulting waveform. Note that pulsar sync is conceptually equivalent to *pulsar synthesis* [15] (hence the name), with the following-oscillator waveform serving as the *pulsaret* in combination with a rectangular *pulsaret window*.

By following the same approach as in Section 2.1 for hard sync, we can derive the complex-valued Fourier-series coefficient transform for the pulsar sync. Again, we assume $T_{\text{follow}} = 1$ and $T_{\text{lead}} = P$. Since we are considering audio signals, we will assume that the following-oscillator waveform has no DC component, so that $c_0 = a_0 = \mathring{a}_0 = 0$. This will simplify some of the resulting expressions. Starting from the real-valued coefficients $\{a_0, \{a_n, b_n\}_{n=1}^N\}$, we first apply the pre-rotation in (5) with $\tau = -1/2$ to get $\{\mathring{a}_0, \{\mathring{a}_n, \mathring{b}_n\}_{n=1}^N\}$. We then derive the spectral-resampling transform for the pulsar sync analogous to the procedure in Section 2.1.2, yielding the coefficients $\{\ddot{a}_0, \{\ddot{a}_n, \ddot{b}_n\}_{n=1}^N\}$ that describe the pulsar-sync signal $\ddot{s}(\cdot)$. The resulting transform for real-valued Fourier-series coefficients is

$$\ddot{a}_0 = \frac{\mathring{a}_0}{P} = 0 \tag{22}$$

$$\ddot{a}_n = \sum_{k=1}^{N} \frac{\mathring{a}_k}{P} \left(\mathrm{sinc}\left(\frac{n}{P} - k\right) + \mathrm{sinc}\left(\frac{n}{P} + k\right)\right) \tag{23}$$

$$\ddot{b}_n = \sum_{k=1}^{N} \frac{\mathring{b}_k}{P} \left(\mathrm{sinc}\left(\frac{n}{P} - k\right) - \mathrm{sinc}\left(\frac{n}{P} + k\right)\right), \tag{24}$$

for $n = 1, \ldots, N$.

In order to arrive at an alias-free output signal $\ddot{s}[\ell]$ for the pulsar sync, we use the same additive-synthesis approach from Section 2.2 with the Fourier-series coefficients $\{\ddot{a}_0, \{\ddot{a}_n, \ddot{b}_n\}_{n=1}^N\}$.

Remark 12. *Although derived for* $P \geq 1$*, the spectral transform (23), (24) appears to extend to* $P < 1$*, where neighboring following-oscillator-waveform cycles overlap, which is similar to* overlapped pulsaret-width modulation *[15, p. 136]. A more detailed analysis of this case is left for future work.*

3. ASIC IMPLEMENTATION

While our described spectral-resampling approach provides an alias-free formulation of oscillator synchronization, it comes with two practical drawbacks: (i) the coefficient transform has a computational complexity of $\mathcal{O}(N^2)$ and (ii) computing the transform coefficients requires evaluating many terms that consist of trigonometric functions and divisions (e.g., sinc or versinc functions). For example, the hard-sync spectral transform (12), (13) with $N = 512$ requires $512^2 \cdot 2 = 524\,288$ sinc function evaluations, each involving a sine function evaluation and a division. For real-time sound synthesis in software with modulated period ratio[3], even a small number of harmonics quickly results in prohibitive complexity. As a reference, an unoptimized NumPy implementation requires between 2.7 ms and 5.8 ms to compute the spectral-resampling transform for a sawtooth following-oscillator waveform and $N = 512$

[2] We chose this terminology in analogy to the sinc function by using the *versed sine* function $\mathrm{versin}(\phi) = 1 - \cos(\phi)$ instead.

[3] Modulation of f_{lead} or f_{follow} implies a modulation of P.

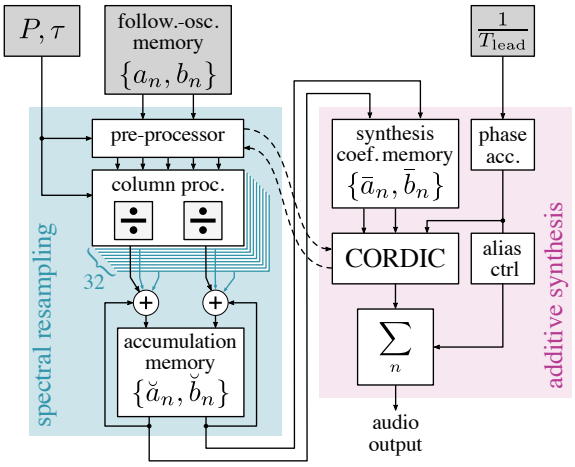


Figure 4: Architecture overview of the HASY ASIC with spectral-resampling engine and additive-synthesis oscillator. Gray elements are user-configurable.

(depending on the sync mode and the period ratio)[4], which corresponds to a hypothetical maximum update rate between 172 Hz and 370 Hz. At these update rates, high modulation rates or depths may introduce audible artifacts each time the Fourier-series coefficients are updated, so a faster computation of the transform is desirable. To this end, we harness the efficiency of ASIC implementations that can be specialized to the specific computations required by our approach. Concretely, we propose an ASIC called *HASY*, which implements the three proposed spectral-resampling methods (hard, mirrored, and pulsar sync) and additive synthesis to enable real-time alias-free oscillator synchronization with $N = 512$ harmonics.

3.1. Architecture Overview

HASY is implemented as custom digital hardware using a fixed-point datapath (bit-widths of order 24 bit) and a parallel architecture to enable real-time execution of the proposed spectral-resampling approach. Figure 4 provides the architecture overview of HASY: The left side of the block diagram depicts the *spectral-resampling engine* that implements the Fourier-coefficient transforms for hard sync, mirrored sync, and pulsar sync described in Sections 2.1, 2.3, and 2.4, respectively. The right side depicts the *additive-synthesis oscillator* that implements the alias-free synthesis of Section 2.2 and is realized as a stripped-down version of the big Fourier oscillator (BFO) [13].

3.1.1. Sound Parameter Inputs

The intended application scenario for HASY is a digital hardware synthesizer system with a central host microcontroller. In this scenario, the host controls HASY via an SPI interface (not shown in Figure 4), which allows the host to set the parameters that determine the sound produced by HASY. Specifically, the SPI interface allows the host to set the following-oscillator Fourier-series coefficients $\{a_n, b_n\}_{n=1}^{512}$ (HASY assumes $a_0 = 0$, which is often reasonable for audio signals as they have no DC component), the leading-to-following-oscillator period ratio P, and the leading-oscillator fundamental frequency f_{lead}, each with 24 bit resolution.

3.1.2. Spectral-Resampling Engine

The spectral-resampling engine consists of four main parts (left side in Figure 4, top to bottom): (i) following-oscillator Fourier-series coefficient memory (short *following-oscillator memory*), (ii) pre-processor, (iii) parallel column processors, and (iv) accumulation Fourier-series coefficient memory (short *accumulation memory*). Those blocks are now discussed in more detail:

(i) The following-oscillator memory contains the real-valued Fourier-series coefficients $\{a_n, b_n\}_{n=1}^{512}$ that are received via SPI.

(ii) When the spectral-resampling engine is running, the *pre-processor* sequentially fetches coefficient pairs $\{a_n, b_n\}$ from the following-oscillator memory and applies the pre-rotation in (5). Besides this, the pre-processor also evaluates trigonometric functions that appear in the spectral-resampling transforms (e.g., $\sin(\pi(n \pm kP))$ for hard sync). These trigonometric functions are computed using a *coordinate rotation digital computer (CORDIC)*[5] [16] that is time-shared with the additive-synthesis side. To further optimize the computation of the trigonometric terms, symmetries of the trigonometric functions are exploited. Ultimately, the pre-processor passes the rotated coefficient pairs $\{\mathring{a}_n, \mathring{b}_n\}$ along with the computed trigonometric values to the parallel column processors.

(iii) An array of 32 parallel *column processors* is used to compute the terms of the $\sum$-sums found in the spectral-resampling transforms. These terms from the spectral-resampling transforms contain quotients (e.g., the fraction that appears in $\operatorname{sinc}(n \pm kP)$ for hard sync). In order to compute these quotients, custom divider units are implemented using the so-called *division-by-hand* algorithm. To avoid numerical issues with very small denominators, the dividers use a zeroth-order or first-order Taylor expansion[6] for denominators smaller than 2^{-12}. The column processors also implement the summation of the quotients (resp. subtraction) that appear in the spectral-resampling transforms. After this, the column processors carry out the multiplication with the input Fourier-series coefficients. Then, the column processors pass the terms of the $\sum$-sums found in the spectral-resampling transforms to the accumulation memory.

(iv) Finally, the *accumulation memory* is dedicated to the accumulation of the terms from the $\sum$-sums in the linear spectral-resampling transform.

3.1.3. Real-Time Capability

HASY is designed to run at a clock frequency f_{clk} that is a fixed multiple of the audio output sample rate f_{s}, i.e., $f_{\text{clk}} = 2048\, f_{\text{s}}$. Thus, for the target audio sample rate of 96 kHz, HASY requires a target clock frequency of $f_{\text{clk}} \approx 196.6$ MHz. Thanks to the parallel architecture of the spectral-resampling engine, which comprises 32 parallel column processors, the Fourier-series coefficient transform for $N = 512$ harmonics is completed in only 8 240 clock cycles, independently of the sync mode. Thus, at the target clock frequency, the spectral-resampling transform takes approximately 42 µs, which

[4] Measured on a MacBook Pro (M2 Pro). Generally, the mirrored-sync resampling transform consistently took longer than the other two sync modes, likely due to the custom versinc function evaluation.

[5] CORDICs allow for efficient yet accurate computation of trigonometric functions.

[6] The hard-sync transform (12), (13) and pulsar-sync transform (23), (24) feature the $\operatorname{sinc}(\cdot)$ quotient, where we use a zeroth-order Taylor approximation. The mirrored-sync transform (18) features $\operatorname{versinc}(\cdot)$, where we use a first-order Taylor approximation.

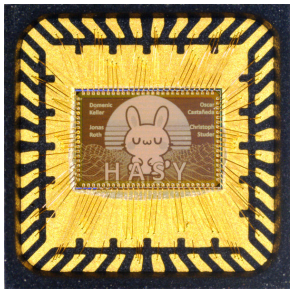


Figure 5: ASIC micrograph. The photo shows the HASY chip inside its package (lid removed). The logo is implemented through the so-called metal fill in the top layer of the chip.

corresponds to just over 4 audio-sample periods at 96 kHz. Thus, if desired, HASY can execute a new spectral-resampling transform every 5 audio samples, which corresponds to a parameter update rate of approximately 19.2 kHz. Such a high parameter update rate makes HASY real-time capable, and modulating the leading oscillator frequency f_{lead} is possible.

3.1.4. Alias-Free Additive-Synthesis Oscillator

As shown on the right side in Figure 4, HASY also contains a stripped-down version of a BFO [13] that synthesizes one voice of the alias-free time-domain audio output signal.

At the end of the spectral-resampling computation, the resulting Fourier-series coefficients are written directly to the *synthesis coefficient memory*. The desired leading-oscillator pitch $1/T_{\text{lead}}$ is set by the system host via SPI. The alias-free additive synthesis in (14) is implemented with a CORDIC that computes the $\sin(\cdot) + \cos(\cdot)$ terms in combination with a conditional accumulation to ensure the Nyquist condition (15) is met. Note that in contrast to (14), the DC component $\bar{a}_0$ is ignored, since it is considered irrelevant for the intended application. The additive synthesizer uses a pipelined architecture with a throughput of one coefficient pair per clock cycle. With $N = 512$ harmonics, the additive synthesis occupies only about one fourth of the available 2048 clock cycles per audio sample. Thus, when the synthesis coefficient memory is receiving newly computed Fourier-series coefficients from the spectral-resampling engine, the additive-synthesis computation is scheduled accordingly to avoid memory-access contention. Finally, the output time-domain signal is passed to an I2S interface with 96 kHz and 24 bit resolution.

In contrast to the BFO ASIC [13], HASY contains only one (mono) oscillator and the number of harmonics is reduced from 1024 to 512. Also, the HASY design operates at a reduced resolution in the datapath from 32 bit to 24 bit to reduce silicon area.

3.2. ASIC Measurement Results and Comparison

Figure 5 shows a micrograph of the 6 mm^2 HASY ASIC, which was fabricated in TSMC 65 nm LP CMOS technology.[7] The ASIC utilizes a QFN40 package and was successfully tested for the hard-sync mode with a sinusoidal following-oscillator waveform with $P = 0.5$ and $f_{\text{lead}} = 3$ kHz. This scenario was also used for the following measurements: At the nominal 1.2 V core supply and 300 K room temperature, the ASIC achieves a maximum measured clock frequency of 250 MHz, exceeding the required 196.6 MHz to support an audio sample rate of $f_{\text{s}} = 96$ kHz. The measured power consumption at 200 MHz clock frequency is 242 mW.

Table 1: Comparison of HASY and BFO ASICs.

	this work HASY	[13] BFO
Oscillator synchronization	yes	no
Voices	1	2
Oscillators per voice	1[a]	4
Harmonics per oscillator	512	1024
Max. clock frequency	250 MHz	154 MHz
Chip area	6 mm^2	3 mm^2
Power consumption	242 mW	178 mW

[a] Our approach to hard sync requires only one oscillator.

Table 1 summarizes the key implementation characteristics of HASY alongside those of the BFO additive oscillator in [13]. Clearly, the support of oscillator synchronization comes at a premium. The BFO chip generates a total of $1024 \cdot 4 \cdot 2 = 8192$ partials per sample in only 3 mm^2, but does not support oscillator synchronization. In contrast, HASY generates 512 harmonics per sample in 6 mm^2, while supporting three distinct oscillator-synchronization modes. This area overhead is almost entirely attributable to the spectral-resampling engine, which accounts for 97% of HASY's cell area, while the additive-synthesis oscillator occupies less than 3%.

We note that the current implementation of HASY contains errors that escaped verification and limit the functionality of the fabricated HASY ASIC. These errors are located in the control logic and affect part of the configuration interface. Nevertheless, the fixed-point golden model together with the ASIC measurements carried out in functioning modes of the chip confirm the core architectural concept and demonstrate that the ASIC achieves real-time processing capability (cf. Table 1). A fully functional reimplementation of the ASIC is ongoing work.

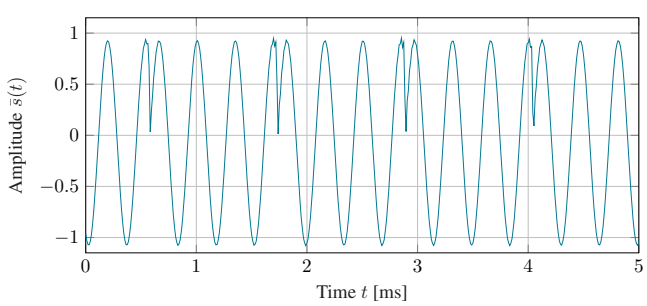
4. EVALUATION

We now evaluate the accuracy of the proposed hard-sync synthesis method and its hardware implementation on the HASY ASIC. For brevity, we restrict our evaluation to the hard-sync mode.

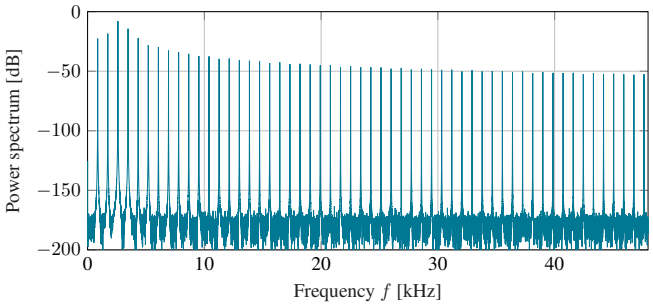
4.1. Baselines and Performance Metric

To assess the accuracy of the proposed ASIC, we use our fixed-point golden model implemented in MATLAB. This model is bit-true to the HASY ASIC, i.e., it replicates the fixed-point arithmetic of the hardware datapath. In what follows, we call this golden model *HASY model*. Figure 6 depicts alias-free hard-sync sine and sawtooth waveforms generated by the HASY model. In addition, Figure 7 shows mirrored sync and pulsar sync applied to unconventional waveforms.

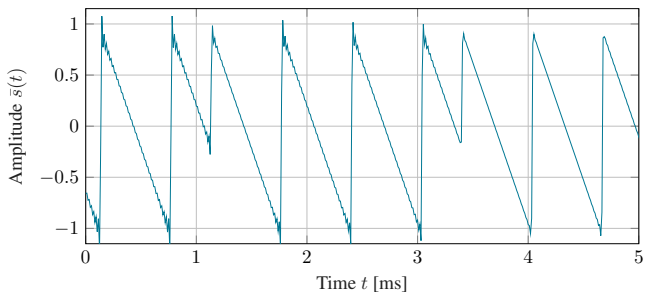
[7] While ASIC design is known to be costly, HASY was fabricated through a multi-project wafer service, where the expensive mask costs are shared between multiple designs. For HASY, 100 chips were fabricated and 10 packaged for a total cost of € 24 000, a process that took five months. The design of the HASY ASIC was enabled by the experience of our research group and the Microelectronics Design Center at ETH Zurich (see the IIS Chip Gallery: `http://asic.ethz.ch`).



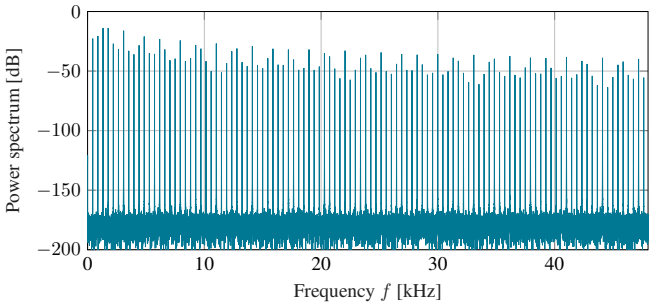
(a) sine hard sync: time domain



(b) sine hard sync: spectrum



(c) sawtooth hard sync: time domain



(d) sawtooth hard sync: spectrum

Figure 6: Hard-sync output generated by the HASY fixed-point golden model in time- (left) and frequency- (right) domains. The sine hard-sync case (top row) uses $f_{\text{follow}} = 2\,900.33\,\text{Hz}$ and $f_{\text{lead}} = 866.42\,\text{Hz}$, as reported in [7, Figure 2]. The sawtooth hard-sync case (bottom row) uses $f_{\text{follow}} = 1\,575\,\text{Hz}$ and $f_{\text{lead}} = 441\,\text{Hz}$, as reported in [8, Figure 6].

For the evaluation, we compare the output signals generated by the HASY model against the following two baselines:

- The first baseline (called "float") is a floating-point reference software implementation that executes the spectral-resampling transform and additive synthesis as described in Section 2, with up to 512 harmonics.
- The second baseline (called "analytical") starts from the ideal hard-sync signal that is not bandlimited. We then analytically compute the Fourier-series coefficients and resynthesize the time-domain signal with up to 512 harmonics.

We generate 1 s of output samples (corresponding to $L = 96\,000$ samples) with the HASY model and the two baselines for different following-oscillator waveforms and different values of P (leading-to-following-oscillator period ratio). Then, we compute the signal-to-noise and distortion ratio (*SINAD*) in the sample domain as follows:

$$\mathit{SINAD} = 10\log_{10}\left(\frac{\sum_{\ell=1}^{L}|\bar{s}[\ell]|^2}{\sum_{\ell=1}^{L}|\bar{s}[\ell]-\tilde{s}[\ell]|^2}\right). \tag{25}$$

Here, $\tilde{s}[\cdot]$ represents the reference signal (i.e., the floating-point reference or the analytical reference).

4.2. SINAD Comparison

Figure 8 shows the *SINAD* for three different following-oscillator waveforms (sine, triangle, and sawtooth) for different values of P. The leading-oscillator frequency is kept at $f_{\text{lead}} = 94\,\text{Hz}$, which corresponds to 510 harmonics below $f_s/2 = 48\,\text{kHz}$, so almost all of the harmonics available to HASY are utilized. From the results in Figure 8, we make the following observations:

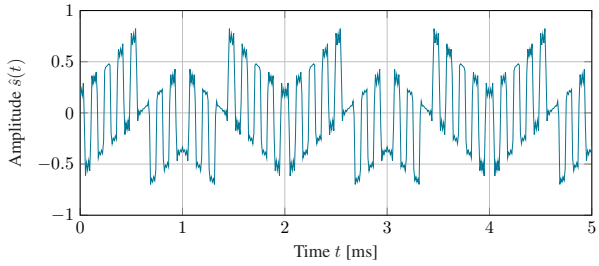
If P is an integer, then the *SINAD* is high, and our approach yields high accuracy. This is not unexpected as for $P \in \mathbb{N}$, the spectral-resampling transform corresponds to a simple Fourier-series coefficient remapping, where the nth harmonic is mapped to the nPth harmonic. If $P < 1$, then the *SINAD* drops. This performance drop is because our method starts from a following-oscillator waveform represented by only 512 Fourier-series coefficients, i.e., already bandlimited (cf. Remark 4). When performing the spectral-resampling transform for $P < 1$, the hard-sync Fourier-series coefficients largely depend on the higher-index Fourier-series coefficients of the following-oscillator waveform. However, since N is fixed to 512, these higher-index Fourier-series coefficients might not be available for resampling. Generally, this effect worsens for smaller values of P and f_{lead}.

Remark 13. *This observation reveals a key limitation of our approach: Resampling can only transform harmonics that are already present in the Fourier-series coefficients of the free-running following waveform.*

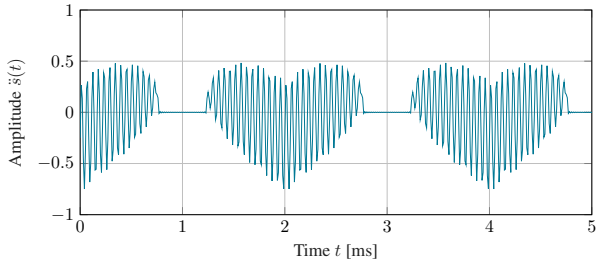
For higher f_{lead} (not shown in Figure 8), the performance drop at small P is less pronounced, since the bandlimited following-oscillator constraint mainly affects higher harmonics (which are no longer within the band of interest at high f_{lead}).

5. CONCLUSIONS

We have introduced a method for alias-free oscillator synchronization based on additive synthesis, which supports three synchronization modes and arbitrary bandlimited following-oscillator waveforms. To address the high computational cost of our approach, we have developed an ASIC, which implements the proposed method



(a) Mirrored-sync sandstorm waveform.



(b) Pulsar-sync heart waveform.

Figure 7: Soft-sync variants applied to unconventional waveforms. The sandstorm waveform is inspired by the one used in Darude's "Sandstorm" [17].

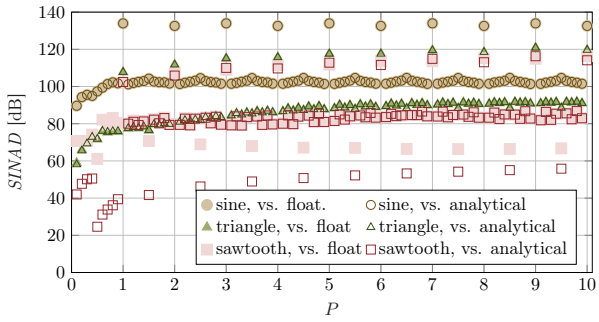


Figure 8: *SINAD* versus P for hard sync of three different following-oscillator waveforms at $f_{\text{lead}} = 94\,\text{Hz}$.

with a latency of fewer than five audio samples, demonstrating practical, real-time realization of our approach. A comparison of the ASIC's output with both an analytical (ideal) and a floating-point baseline reveals that our implementation achieves *SINAD* values above 41 dB (if the following oscillator's frequency exceeds that of the leading oscillator).

There are several avenues for future work. First, we are working on a polyphonic version of the HASY ASIC that also fixes its implementation bugs. Second, for certain waveforms, Fourier-series coefficients could be derived directly from the analytical synchronized functions. Finally, we are developing a Eurorack module to demonstrate the HASY ASIC in a practical instrument.

On the companion page of this article we provide a reference Python (NumPy) implementation and generated audio examples.[8]

[8] `https://github.com/IIP-Group/hasy-python`

6. ACKNOWLEDGMENTS

The authors thank the Microelectronics Design Center of ETH Zurich for their help with the tape-out of the HASY ASIC.

7. REFERENCES

[1] Breakwater. (1980, Apr.) Release the beast. Accessed: Apr. 3, 2026. [Online]. Available: https://www.youtube.com/watch?v=KdBV3c5Vpz8.

[2] H. Faltermeyer. (1985, May) Fletch theme. Accessed: Apr. 3, 2026. [Online]. Available: https://www.youtube.com/watch?v=r-lu0vjHSO0.

[3] E. Brandt, "Hard sync without aliasing," in *Proc. Int. Comput. Music Conf. (ICMC)*, Sep. 2001.

[4] V. Välimäki and A. Huovilainen, "Antialiasing oscillators in subtractive synthesis," *IEEE Signal Process. Mag.*, vol. 24, no. 2, pp. 116–125, 2007.

[5] T. Stilson and J. Smith, "Alias-free digital synthesis of classic analog waveforms," in *Proc. Int. Comput. Music Conf. (ICMC)*, Aug. 1996, pp. 332–335.

[6] J. Nam, V. Välimäki, J. S. Abel, and J. O. Smith, "Efficient antialiasing oscillator algorithms using low-order fractional delay filters," *IEEE Trans. Audio Speech Lang. Process.*, vol. 18, no. 4, pp. 773–785, 2009.

[7] P. P. La Pastina and S. D'Angelo, "A general antialiasing method for sine hard sync," in *Proc. Int. Conf. Digital Audio Effects (DAFx22)*, Sep. 2022, pp. 109–114.

[8] J. Timoney, V. Lazzarini, M. Hodgkinson, J. Kleimola, J. Pekonen, and V. Välimäki, "Virtual analog oscillator hard synchronisation: Fourier series and an efficient implementation," in *Proc. Int. Conf. Digital Audio Effects (DAFx-12)*, Sep. 2012.

[9] J. Kleimola and V. Välimäki, "Reducing aliasing from synthetic audio signals using polynomial transition regions," *IEEE Signal Process. Lett.*, vol. 19, no. 2, pp. 67–70, 2011.

[10] H. Chamberlin, *Musical Applications of Microprocessors*, 2nd ed. Rochelle Park, NJ: Hayden Books, 1985.

[11] X. Rodet and P. Depalle, "Spectral envelopes and inverse FFT synthesis," in *Proc. Audio Eng. Soc. Conv. 93*, Oct. 1992.

[12] F. De Bernardinis, R. Roncella, R. Saletti, P. Terreni, and G. Bertini, "An efficient VLSI architecture for real-time additive synthesis of musical signals," *IEEE Trans. VLSI Syst.*, vol. 7, no. 1, pp. 105–110, Mar. 1999.

[13] J. Roth, D. Keller, O. Castañeda, and C. Studer, "An aliasing-free hybrid digital-analog polyphonic synthesizer," in *Proc. Int. Conf. Digital Audio Effects (DAFx23)*, Sep. 2023.

[14] G. Deslauriers and C. Leider, "A bandlimited oscillator by frequency-domain synthesis for virtual analog applications," in *Proc. Audio Eng. Soc. Conv. 127*, Oct. 2009.

[15] C. Roads, "Sound composition with pulsars," *J. Audio Eng. Soc.*, vol. 49, no. 3, pp. 134–147, Mar. 2001.

[16] J. E. Volder, "The CORDIC trigonometric computing technique," *IRE Trans. Electron. Comput.*, vol. EC-8, no. 3, pp. 330–334, Sep. 1959.

[17] Darude. (1999, Oct.) Sandstorm. Accessed: Apr. 3, 2026. [Online]. Available: https://www.youtube.com/watch?v=y6120QOlsfU.