AN ALIASING-FREE HYBRID DIGITAL-ANALOG POLYPHONIC SYNTHESIZER

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

Analog subtractive synthesizers are generally considered to provide superior sound quality compared to digital emulations. However, analog circuitry requires calibration and suffers from aging, temperature instability, and limited flexibility in generating a wide variety of waveforms. Digital synthesis can mitigate many of these drawbacks, but generating arbitrary aliasing-free waveforms remains challenging. In this paper, we present the $\pm$synth, a hybrid digital-analog eight-voice polyphonic synthesizer prototype that combines the best of both worlds. At the heart of the synthesizer is the big Fourier oscillator (BFO), a novel digital very-large scale integration (VLSI) design that utilizes additive synthesis to generate a wide variety of aliasing-free waveforms. Each BFO produces two voices, using four oscillators per voice. A single oscillator can generate up to 1024 freely configurable partials (harmonic or inharmonic), which are calculated using coordinate rotation digital computers (CORDICs). The BFOs were fabricated as 65 nm CMOS custom application-specific integrated circuits (ASICs), which are integrated in the $\pm$synth to simultaneously generate up to 32 768 partials. Four 24-bit 96 kHz stereo DACs then convert the eight voices into the analog domain, followed by digitally controlled analog low-pass filtering and amplification. Measurement results of the $\pm$synth prototype demonstrate high fidelity and low latency.

1. INTRODUCTION

Digital sound synthesis has many advantages over implementations with analog circuitry. Most notably, digital implementations are able to produce a wide variety of waveforms and do not suffer from temperature instabilities, aging, and component variations. However, aliasing is a ubiquitous nuisance in digital sound synthesis and specialized signal processing techniques are often necessary to combat such artifacts. For example, the work in [1] proposes low-complexity methods to generate aliasing-free waveforms of classical analog synthesizers (e.g., rectangle, sawtooth, and triangle). Nonetheless, this method is unable to generate more complex waveforms. In stark contrast, direct digital synthesis (DDS) [2] enables the generation of nearly arbitrary waveforms at very low complexity. Unfortunately, naïve DDS implementations generally suffer from aliasing. While aliasing can be reduced to a certain extent with oversampling followed by low-pass filtering, such an approach diminishes the complexity advantages of DDS. The work in [3] presents a method that is able to generate arbitrary alias-free single-period wavetable waveforms. This method, however, requires intricate trigonometric functions that must be calculated at high precision and is, thus, not well-suited for efficient software and hardware implementations.

An alternative sound-synthesis approach that eliminates aliasing altogether is to use additive synthesis [4], without ever generating partials that exceed half the sampling rate. Many software synthesizers support additive synthesis and benefit from the flexibility and user-interface capabilities that software brings. However, relying on general-purpose processors, such implementations have to balance signal quality and computational complexity, which limits the amount of partials that can be generated and affects their purity.[1] To overcome the limitations of additive synthesis software implementations, the work in [6] proposes a specialized hardware design, which is able to generate a large number of partials (up to 1200) with a single application-specific integrated circuit (ASIC). Such an implementation would enable the generation of a wide variety of complex and aliasing-free waveforms, but, to the best of our knowledge, no working system was demonstrated.

In recent years, a number of commercially available hybrid digital-analog instruments emerged, which can generate a broad range of high-quality waveforms. Specific instances are the Arturia Freak Series [7], Sequential Prophet X [8], Udo Super 6 [9], and Waldorf Quantum [10]. Unfortunately, only very little is known about the inner workings of these instruments and, thus, it remains largely a mystery how the waveforms are synthesized.

1.1. Contributions

We present the $\pm$synth, an eight-voice polyphonic hybrid digital-analog synthesizer prototype that combines digital oscillators with analog filtering and amplification. Each voice consists of four digital oscillators, each able to generate a wide range of aliasing-free waveforms using an additive synthesis approach[2] with up to 1024 partials (harmonic, inharmonic, or subharmonic); in total, the instrument can generate 32 768 partials simultaneously. The oscillators are implemented using a custom ASIC, the *big Fourier oscillator (BFO)*, which generates two voices. To arrive at high hardware efficiency of the BFO ASIC implementation, we present a range of algorithm-level optimizations and a very-large scale integration (VLSI) architecture that utilizes CORDICs (short for coordinate rotation digital computers) to generate partials of high purity. We show how the BFO ASICs are integrated into the $\pm$synth hardware prototype, including the analog section that incorporates voltage controlled filters (VCFs) and voltage controlled amplifiers (VCAs) based on commercial integrated circuits (ICs). Finally, we

[1]The maximum number of partials varies significantly across different plugins and depends on several parameters, such as the number of voices, signal quality, and others. Alchemy from Apple's Logic Pro X, for example, supports up to 600 partials [5], while others support less than a dozen.

[2]Thus the name $\pm$*synth*, where $+$ represents the additive synthesis approach and $-$ the subtractive architecture of the instrument.

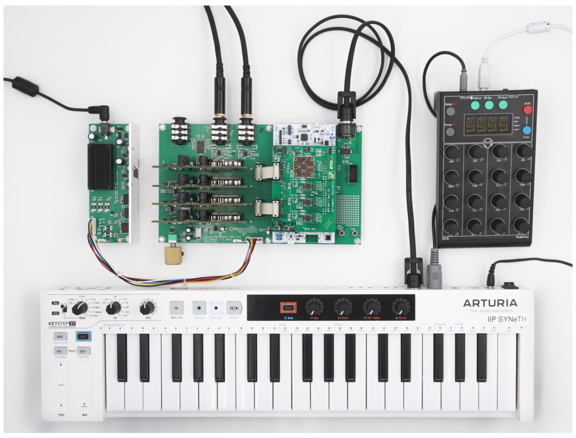


Figure 1: *Overview of the synthesizer setup. Top left: custom power supply; top middle: $\pm$synth hardware prototype; top right: MIDI controller; bottom: MIDI keyboard.*

present implementation results of the BFO ASIC and measurement results at various output stages in the instrument. A photo of the $\pm$synth hardware prototype, including the external MIDI keyboard and controller, is shown in Figure 1.

2. SYNTHESIZER ARCHITECTURE

Figure 2 shows a photo of the $\pm$synth hardware prototype consisting of a main board (with audio, MIDI, and power connectors), an STM32 Nucleo development board (in white), with a custom printed circuit board (PCB) on top hosting four BFO ASICs, and four analog voice PCBs plugged into the main board (top right). The key components of the hardware design are discussed next.

2.1. System Overview

A system architecture overview of the $\pm$synth is given in Figure 3. The instrument relies on a hybrid digital-analog version of subtractive synthesis with digital oscillators, analog filters, and analog amplifiers. The $\pm$synth is digitally controlled by a microcontroller unit (MCU), which receives user inputs from an external MIDI keyboard and a universal MIDI controller with rotary encoders. The MCU generates all of the control signals for the digital oscillators as well as for the analog filters and amplifiers. The instrument is able to generate eight independent voices, where each BFO ASIC implements two voices and each voice consists of four digital oscillators. The digital voices are converted into the analog domain using four stereo DACs and each voice is separately processed by a VCF and a VCA; an *analog voice PCB* houses these analog components required for two voices. The control voltages (CVs) are generated using CV-DACs, which are digitally controlled by the MCU. The analog voices are summed to create the instrument's output signal, which can be either mono (eight voices) or stereo (four voices per channel); this is controlled by relays. The $\pm$synth offers stereo line-level and headphone outputs. A custom power supply derives all of the required voltages for both the analog and digital domains from an off-the-shelf 24 V DC power supply.

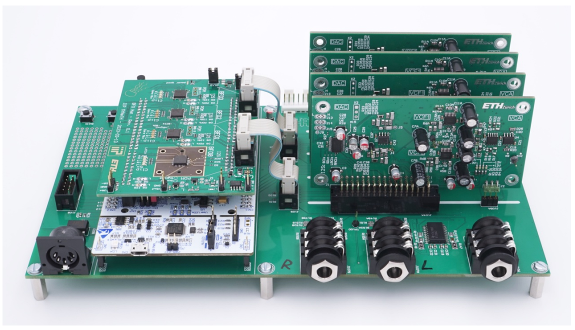


Figure 2: *Photo of the $\pm$synth hardware prototype.*

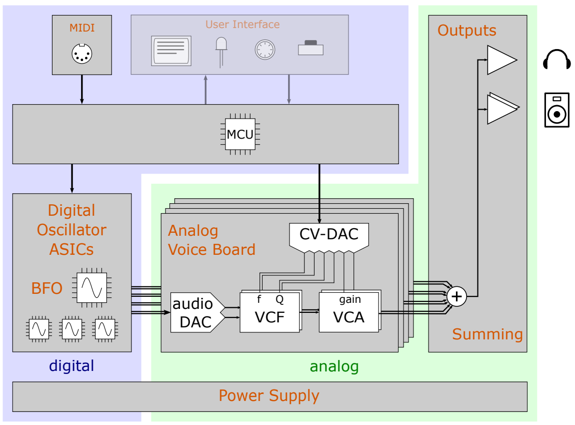


Figure 3: *Architecture overview of the $\pm$synth.*

2.2. Digital Control and Interfaces

Digital control of the instrument is carried out on an STM23 Nucleo development board (STM32F722ZE), which was chosen to bypass the recent chip shortage. Received MIDI messages are used to compute the control signals for the audio path (e.g., oscillator pitch and volume, filter cutoff-frequency and resonance, and amplifier gain). The MCU also implements the attack, decay, sustain, and release (ADSR) envelope generators as well as low-frequency oscillators (LFOs). The MCU generates and transmits the control signals to the digital oscillators using a serial parallel interface (SPI) bus. The oscillators then generate eight voices that are converted to the analog domain via inter-IC sound (I2S) interfaces and four 24-bit stereo DACs. A second SPI bus on the MCU interfaces with the CV-DACs that control the analog voice processing paths. The details of the digital oscillators and analog voice PCBs are discussed next.

2.3. Aliasing-Free Digital Oscillator

At the heart of the $\pm$synth are digital aliasing-free oscillators, which utilize additive synthesis, each generating up to $K = 1024$ partials.[3] Each voice is composed of four digital oscillators, which can be mixed together arbitrarily with configurable gains. Our custom ASIC, the BFO, implements two voices, and the $\pm$synth consists of four BFO ASICs to provide eight-voice polyphony. Each oscillator

[3]With up to $K = 1024$ partials per oscillator, we can generate, for example, a sawtooth wave with a base frequency $f = 20$ Hz and the highest harmonic at 20 480 Hz, which is at the edge of the audible spectrum.

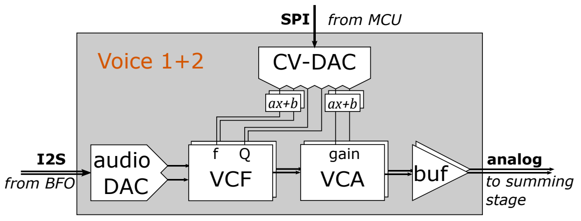


Figure 4: *Block diagram of an analog voice PCB.*

calculates its samples $x[\ell]$ using the following Fourier series:

$$x[\ell] = \sum_{k=1}^{K} a_k \cos\left(2\pi \frac{f n_k}{f_s}\ell\right) + b_k \sin\left(2\pi \frac{f n_k}{f_s}\ell\right). \quad (1)$$

The oscillator parameters $\{a_k, b_k, n_k\}_{k=1}^{K}$, together with the oscillator's base frequency f and the system's sampling rate f_s, fully determine the waveform to be generated. Note that each oscillator has its own set of parameters. In order to avoid aliasing altogether, we only sum the terms in (1) indexed by the set[4]

$$\mathcal{K}(f) = \{k = 1, \dots, K : f n_k < f_s/2\}, \quad (2)$$

i.e., we replace $k = 1, \dots, K$ with $k \in \mathcal{K}(f)$ in (1). Section 3 details how this additive synthesis approach is implemented in our custom BFO ASICs.

Remark 1. *We emphasize that (1) is, strictly speaking, not a Fourier series, as we also allow the multipliers $n_k \in \mathbb{Q}$, $k = 1, \dots, K$, to be nonnegative rational numbers represented in the chosen fixed-point format (see Section 3.1 for the details). This flexibility enables us to generate waveforms with harmonics, inharmonics, and subharmonics, which implies that a single oscillator cannot only generate standard analog synthesizer waveforms, but also arbitrary wavetable sounds or bell-like timbres.*

Each of the four oscillators of a voice generates samples according to (1), which are weighted and summed. Then, the two BFO voices can be further mixed before being passed to the I2S output. Details on the mixing stage are provided in Section 3.5.

2.4. Analog Voice Boards

The ±synth features four analog voice PCBs, which can be seen at the top-right of Figure 2. Each of these PCBs carries out analog signal processing for two voices; a block diagram is depicted in Figure 4. The audio DAC receives the samples from the two voices generated by a BFO ASIC at a sampling rate of 96 kHz. We used a CS4350 24-bit stereo DAC from Cirrus Logic [11], which contains an integrated phase-locked loop (PLL) that derives its master clock from the I2S clock. This eliminates the need to route a separate 25 MHz clock signal to each analog voice PCB.

The VCFs are implemented using the SSI2144 IC from Sound Semiconductor [12], which implements a fourth-order low-pass ladder filter. The cutoff frequency and resonance (Q-factor) are set by CVs. The VCAs are implemented using the SSI2162 IC from Sound Semiconductor [13], which is used to apply the envelope determined by another CV. Finally, the analog voice signal is buffered before leaving the PCB to the final summing stage on the main board that produces headphone and line outputs. Note that some CVs are low-pass filtered, scaled, and offset in order to avoid crosstalk to the audio path and to match the required voltage range; this is implemented using operational amplifiers.

[4]We note that aliasing can still occur if one reconfigures the oscillator parameters $\{a_k, b_k, n_k\}_{k=1}^{K}$ at too fast rates.

3. VLSI DESIGN OF THE BIG FOURIER OSCILLATOR

In order to develop an efficient VLSI design that is able to implement multiple aliasing-free oscillators, each with a large number of high-quality partials, we leverage a range of algorithm- and hardware-level tricks which are discussed next.

3.1. Fixed-Point Arithmetic

Our VLSI design exclusively uses fixed-point arithmetic, mostly with 32-bit fixed-point precision, which enables the generation of extremely pure partials with low harmonic distortion at high hardware efficiency. The fixed-point number format is designated by the notation $\{\square, q_\text{i}, q_\text{f}\}$, where q_i is the number of integer bits, q_f is the number of fractional bits, and $\square$ is either s or u if the fixed-point number is signed or unsigned, respectively. The coefficients a_k and b_k in (1) use the format $\{s, 0, 31\}$, which gives precise control over the partials' amplitudes and phases. The multipliers n_k use the format $\{u, 16, 16\}$, which enables harmonic, inharmonic, and subharmonic partials with fine frequency resolution. The base frequency is normalized as f/f_s and uses the format $\{u, 0, 32\}$, which results in a frequency resolution of 22.4 µHz at $f_s = 96$ kHz. While the generated samples have an internal resolution of 32 bits, the I2S output is reduced to the DACs' resolution of 24 bits.

3.2. Algorithm-Level Optimizations

To improve the hardware efficiency of our VLSI design, we use the following algorithm-level optimizations that reduce the complexity of calculating the samples as in (1).

Reparametrization from Radians to Turns Instead of directly computing the arguments $\phi_k[\ell] \triangleq 2\pi \frac{f n_k}{f_s}\ell$ of the cosine and sine functions in (1), which are in the unit of radians, our VLSI design calculates the two functions $\text{co}(\theta) \triangleq \cos(2\pi\theta)$ and $\text{si}(\theta) \triangleq \sin(2\pi\theta)$ instead. Here, the arguments $\theta_k[\ell] \triangleq \frac{f n_k}{f_s}\ell$ are represented in what is known as *turns*, which has several advantages. First, the arguments $\theta_k[\ell]$ are in the range $[0, 1)$, which requires one to pass arguments to the $\text{co}(\cdot)$ and $\text{si}(\cdot)$ functions of the form

$$\theta_k[\ell] = \frac{f n_k}{f_s}\ell \bmod 1. \quad (3)$$

The modulo-1 operation can be obtained in hardware for free by simply discarding the integer part of $\theta_k[\ell]$ when represented by unsigned fixed-point numbers. Second, one can directly calculate the functions $\text{co}(\cdot)$ and $\text{si}(\cdot)$ in a hardware-friendly manner using CORDICs; see Section 3.3 for the details.

Sequential Calculation of Arguments For each sample $\ell = 0, 1, \dots$ and partial $k = 1, \dots, K$, two multiplications are required to calculate the argument $\theta_k[\ell]$ in (3). First, the argument increment $\delta_k \triangleq f n_k / f_s$ is computed by multiplying the normalized base frequency f/f_s by the frequency multiplier n_k. Second, the argument increment δ_k is multiplied by the sample index ℓ. While this last multiplication occurs in modulo-1 arithmetic, it still requires a high dynamic range, as the sample index ℓ is represented with a large number of bits to avoid unwanted resetting in the oscillators. Hence, every sample ℓ requires $K = 1024$ of these high-resolution $\theta_k[\ell] = \delta_k \ell \bmod 1$ products. This complexity could easily be reduced by tracking the argument θ_k for each partial $k = 1, \ldots, K$ with an addition instead of a multiplication. In specific, one can update the argument θ_k for every new sample ℓ as

$$\theta_k \leftarrow (\theta_k + \delta_k) \bmod 1. \tag{4}$$

The arguments are initialized as $\theta_k = 0$ at sample index $\ell = 0$. We reiterate that the modulo-1 operation is free in hardware by discarding the integer part of the arguments θ_k, $k = 1, \ldots, K$.

Single Argument for All Partials The remaining disadvantage of the above method is that one must keep track of the arguments θ_k for every partial $k = 1, \ldots, K$. This requires additional storage for all arguments in a two-port memory that supports one simultaneous read and write per update of (4). To avoid an additional memory, we keep track of a *single* base-frequency argument $\theta \triangleq \frac{f}{f_s}\ell$ from which all arguments θ_k can be derived as follows:

$$\theta_k = \theta n_k \bmod 1, \quad k = 1, \ldots, K. \tag{5}$$

Unfortunately, unlike the arguments θ_k in (4), the base-frequency argument θ cannot be accumulated modulo-1 since the property

$$\theta n_k \bmod 1 = (\theta \bmod m) n_k \bmod 1 \tag{6}$$

with $m = 1$ does *not* hold for every n_k. For example, for $\theta = 1$ and $n_k = 1.5$, $(\theta \bmod 1)n_k \bmod 1 = 0$ is different from the desired $\theta n_k \bmod 1 = 0.5$. Indeed, to calculate θ_k for an arbitrary frequency multiplier n_k from the base-frequency argument θ, the quantity θ needs to be represented with an infinite number of integer bits (i.e., without using a modulo operation). Nevertheless, provided that the multipliers n_k are represented with a finite number of fractional bits q_f, it is sufficient to represent θ using a finite number of integer bits. This key insight is made rigorous by Lemma 1.

Lemma 1. *$\theta n \bmod l = (\theta \bmod m)n \bmod l$ if $mn \bmod l = 0$.*

Proof. Let $\Theta = \theta \bmod m$, so that we want to show $\theta n \bmod l = \Theta n \bmod l$. We rewrite $\Theta = \theta \bmod m$ and $mn \bmod l = 0$ as

$$\theta = am + \Theta, \text{ and} \tag{7}$$
$$mn = bl, \tag{8}$$

where $a, b \in \mathbb{Z}$. Multiplying both sides of (7) by n, we get

$$\theta n = amn + \Theta n = abl + \Theta n, \tag{9}$$

where (9) follows from (8). Since $ab \in \mathbb{Z}$, taking the modulo-l of both sides of (9) results in $\theta n \bmod l = \Theta n \bmod l$. □

By applying Lemma 1 with $n = n_k$ and $l = 1$, we can determine the value of m so that (6) is satisfied for the multipliers n_k used in the BFO. In words, Lemma 1 is telling us that, instead of keeping track of θ with infinite precision, we can just keep track of its modulo-m equivalent and any multiplication by the frequency multipliers n_k will be correct in modulo-1 arithmetic as long as $mn_k \in \mathbb{Z}$. Given that n_k has $q_\text{f} = 16$ fractional bits, we can satisfy this last requirement by setting $m = 2^{q_\text{f}}$. Therefore, we keep track of the base-frequency argument θ with the recursion

$$\theta \leftarrow (\theta + \delta) \bmod 2^{q_\text{f}}, \tag{10}$$

where $\delta = f/f_s$. The modulo-2^{q_f} operation is easily implemented in hardware by using only q_f bits to represent the integer part of θ and letting the result wrap-around once the maximum representable number is reached. We note that, if all $n_k \in \mathbb{Z}$, then θ could be tracked in modulo-1 arithmetic—thus, the recursion in (10) is required as we also support inharmonic and subharmonic partials.

We observe that, while this approach requires $K = 1024$ multiplications per sample ℓ as in (5), it avoids (i) storing arguments per partial and (ii) additional $K = 1024$ multiplications per sample that would be required to compute the frequency increments $\delta_k = \delta n_k$. Thus, our approach leverages a trade-off between the high complexity of naïvely computing $\theta_k[\ell]$ as in (3) and the large memory overhead of a per-θ_k-argument accumulation as in (4).

3.3. Computing Cosines and Sines with CORDICs

To calculate cosine and sine functions at high precision and in a hardware-friendly way, we utilize CORDICs [14], which essentially calculate two-dimensional Givens rotations of the following form:

$$\underbrace{\begin{bmatrix} p'_1 \\ p'_2 \end{bmatrix}}_{\mathbf{p}'} = \underbrace{\begin{bmatrix} \cos(\phi) & -\sin(\phi) \\ \sin(\phi) & \cos(\phi) \end{bmatrix}}_{=\mathbf{G}(\phi)} \underbrace{\begin{bmatrix} p_1 \\ p_2 \end{bmatrix}}_{=\mathbf{p}}. \tag{11}$$

Evidently, by setting $\phi = 2\pi \frac{f n_k}{f_s} \ell$, $p_1 = a_k$, and $p_2 = -b_k$, the output p'_1 is exactly one term of the Fourier series in (1).

We now outline the idea behind CORDICs—the interested reader is referred to [15] for more details. First, one approximates the desired rotation angle $\phi \approx \sum_{m=0}^{M-1} \phi_m$ by summing M predefined micro-rotation angles ϕ_m, $m = 0, \ldots, M-1$; see below for a concrete choice of these angles. Second, the Givens rotation in (11) is approximated by $\mathbf{G}(\phi) \approx \prod_{m=0}^{M-1} \mathbf{G}(\phi_m)$ with M so-called micro-rotations $\mathbf{G}(\phi_m)$, which are simply Givens rotations by the angles ϕ_m. Third, one rewrites each micro-rotation as

$$\mathbf{G}(\phi_m) = \kappa_m \begin{bmatrix} 1 & -\tan(\phi_m) \\ \tan(\phi_m) & 1 \end{bmatrix}, \tag{12}$$

where $\kappa_m = (1 + \tan^2(\phi_m))^{-\frac{1}{2}}$. Fourth, one restricts the micro-rotation angles ϕ_m to $\tan(\phi_m) = d_m 2^{-m}$ with $d_m \in \{-1, +1\}$. With this, the Givens rotation in (11) is approximated as

$$\mathbf{p}' \approx \kappa \prod_{m=0}^{M-1} \begin{bmatrix} 1 & -d_m 2^{-m} \\ d_m 2^{-m} & 1 \end{bmatrix} \mathbf{p}. \tag{13}$$

Here, the scaling factor $\kappa = \prod_{m=0}^{M-1} \kappa_m$ depends only on the number of micro-rotations M and not on the choices of d_m. Fifth, one needs to determine the micro-rotation angles ϕ_m that well-approximate the target angle ϕ. The standard procedure iteratively determines ϕ_m from the target angle ϕ, i.e., by first taking the angle $\phi_0 = d_0 \text{atan}(2^{-0})$ that brings ϕ closer to zero. One then updates the target angle as $\phi \leftarrow \phi - d_0 \text{atan}(2^{-0})$ and takes a new angle $\phi_1 = d_1 \text{atan}(2^{-1})$ that brings the updated ϕ closer to zero. This procedure is repeated for the remaining $M - 2$ micro-rotations.

We note that every additional micro-rotation provides roughly one additional bit of precision [15]; this implies that the approximation error in (13) can be made arbitrarily small. In our application, we use a quadrant correction followed by $M = 26$ micro-rotations, which leads to a precision of $q_\text{f} = 24$ fraction bits (see Section 4 for measurements). Furthermore, instead of representing the micro-rotation angles ϕ_m in radians, we represent them in turns; this

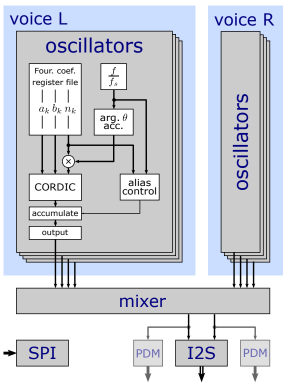


Figure 5: *BFO architecture.*

Figure 6: *ASIC micrograph.*

enables us to directly calculate the functions $\text{co}(\cdot)$ and $\text{si}(\cdot)$ with a CORDIC. Finally, it is crucial to realize that each micro-rotation in (13) only involves shifts, additions, subtractions, and a multiplication by the constant κ; this implies that the Givens rotation in (11) can be approximated in a hardware-friendly manner, generating samples of cosine and sine functions with extremely high purity.

3.4. VLSI Architecture

Figure 5 depicts the VLSI architecture of our BFO ASIC, which consists of two voices, referred to as L and R, each one with four oscillators. Each oscillator features a 1024×96-bit register file to store the oscillator coefficients $\{a_k, b_k, n_k\}$, $k = 1, \ldots, 1024$, an argument accumulator, a fully unrolled CORDIC module that calculates one pair of sine and cosine values per clock cycle, and a sample accumulator. The configuration registers and register files of the BFO are addressed using a memory map and are configured via SPI. Each SPI command uses 48 bits: 16 bits for the address and 32 bits for the data. The SPI interface runs at a baud rate of 13 Mb/s, with which a new waveform with $K = 1024$ partials could be reprogrammed in less than 12 ms—nevertheless, this task is currently completed in 315 ms as our firmware does not yet fully exploit the MCU's capabilities.

To generate samples $x[\ell]$ as in (1) with $K = 1024$ partials at a sampling rate $f_\text{s} = 96$ kHz, each oscillator operates at a clock frequency of $Kf_\text{s} = 98.304$ MHz and calculates one sample of one partial every clock cycle. The argument accumulator uses the base frequency f/f_s, stored in a configuration register, to track the base-frequency argument θ, which is updated every $K = 1024$ clock cycles as in (10). In each of the K clock cycles that the base-frequency argument θ remains fixed, one word of the register file is read to obtain $\{a_k, b_k, n_k\}$. Then, the argument θ_k is computed by multiplying θ and n_k as in (5). With $\{a_k, b_k, \theta_k\}$ available, the CORDIC computes one partial in (1), which is then accumulated to the current sample $x[\ell]$ if aliasing will not occur, i.e., if $fn_k < f_\text{s}/2$ is met. After $K = 1024$ clock cycles, the four samples generated by the four oscillators are added to create one voice sample, and the L and R voice samples are then mixed using a programmable 2×2 matrix to support, e.g., stereo and mono processing. The two mixed samples are then streamed to the DACs via I2S.

Figure 6 shows a micrograph of the 3 mm^2 BFO ASIC, which was fabricated in TSMC 65 nm LP CMOS technology. At the nominal 1.2 V core supply and room temperature, the ASIC achieves a maximum measured clock frequency of 154 MHz, exceeding the required 98.304 MHz, while consuming only 178 mW.

3.5. Bells and Whistles

The BFO includes a number of additional features, which further improve its versatility and flexibility. These features are as follows.

Subwave Mixing Per default, each oscillator accumulates 1024 partials together to compute one sample $x[\ell]$ as in (1). We also support a *subwave mixing* mode, in which the 1024 partials are split into up to four disjoint groups (or *subwaves*) that are accumulated independently. By doing so, a single oscillator can blend various wavetable sounds with the same base frequency, each one with fewer partials; e.g., an oscillator can generate four subwaves of 256 partials each instead of a single waveform with 1024 partials. Each subwave $x_i[\ell]$, $i = 1, \ldots, 4$, has a corresponding weight v_i, so that the output of one oscillator is $y[\ell] = \sum_{i=1}^{4} v_i x_i[\ell]$. Thus, with the four oscillators per voice, subwave mixing can arbitrarily blend up to 16 different wavetables in a single voice.

Aliasing Control Per default, only partials for which $fn_k/f_\text{s} < 0.5$ holds are accumulated; see (2). The BFO includes a mode that accumulates partials for which the following condition is met

$$f_\text{HP} \leq fn_k/f_\text{s} < f_\text{LP}, \tag{14}$$

which realizes an ideal band-pass filter with the lower and upper cutoff frequencies f_HP and f_LP, respectively. These frequencies can be configured in the range $[0, 1)$; the default values are $f_\text{HP} = 0$ and $f_\text{LP} = 0.5$ (no aliasing allowed). Having these frequencies programmable can be used, e.g., to intentionally allow for aliasing (by setting $f_\text{LP} > 0.5$), or to apply low- and high-pass filtering.

Bit- and Rate-Crusher Each oscillator output includes (optional) *bit-crusher* and *rate-crusher* distortion effects. The bit-crusher forces certain bits of the output sample $y[\ell]$ to zero; the zero bits are determined by a programmable bit-mask. This feature can be used to reduce the bit-resolution by zeroing a certain number of least-significant bits; other, more complicated masking patterns are also possible. The rate-crusher is a simple sample-and-hold sub-sampling circuit that lowers the rate at which the output samples $y[\ell]$ are updated. The sub-sampling rate is set in the range $(0, f_\text{s})$ and is stored in a configuration register. This feature can be used to emulate lower sampling rates or, for example, cause intriguing aliasing artifacts that depend on the base frequency f.

PDM Output As an alternative to the I2S interface, each BFO ASIC also includes two pulse density modulation (PDM) outputs. The 1-bit PDM signal is generated from a digital first-order sigma-delta modulator running at an oversampling rate of 1024, which corresponds to the BFO's clock frequency. This output could be useful for cost-sensitive applications (as no DACs are required), but at reduced sound quality; see Section 4.1 for measurement results. Note that the PDM outputs are not used in the $\pm$synth prototype.

Clipping Indicators At certain stages in the digital signal processing path, sample values must be clipped to a certain maximum range to reduce their word lengths. This occurs at the output of each oscillator and after the mixer stage. Thus, we included a number of clipping indicator outputs that are raised if clipping occurred; these are tied to LEDs on the $\pm$synth prototype.

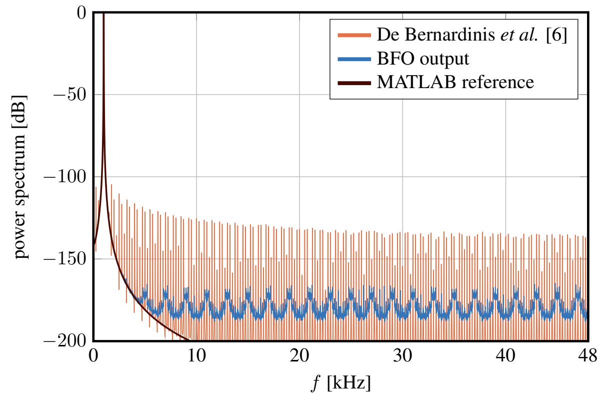


Figure 7: *Power spectrum comparison for a* 1 *kHz sine wave at a sampling rate of* $f_s = 96$ *kHz between a MATLAB floating-point reference, the BFO output, and the method described in [6].*

3.6. Comparison

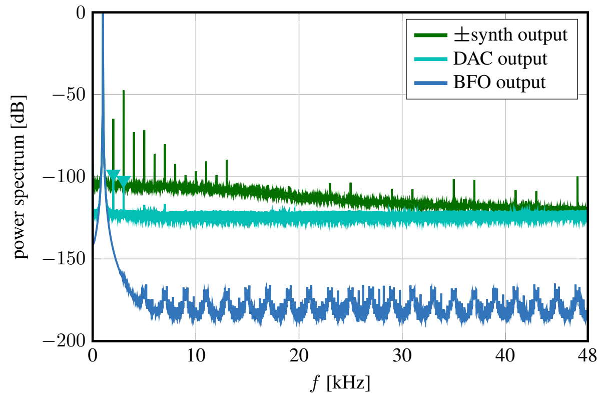
To the best of our knowledge, only the ASIC design reported in [6] is comparable to our BFO implementation. The design in [6] utilizes a marginally stable infinite-impulse response (IIR) filter to recursively generate samples of a single sinusoid only with the help of a multiply-accumulate unit. While the algorithm itself is competitive to our approach in complexity per generated sample, the recursive calculations together with fixed-point arithmetic suffer from error propagation, particularly at low frequencies (in the order of tens of Hz). To mitigate this issue, the recursion must be restarted periodically (the authors recommend restarting every 128 samples), which requires one to retrieve two sinusoids from two consecutive sample instants from a memory or a CORDIC. The design in [6] assumes that these initial sine values are generated externally. Moreover, their circuit assumes that the magnitudes and phases are stored externally and streamed into the ASIC. Thus, their design would require additional (external) logic and memory, whereas our BFO ASIC is fully self-contained.

A direct comparison between the hardware implementation characteristics of our BFO ASIC and the design in [6] is challenging due to missing details. Nonetheless, we re-implemented their method in MATLAB using the fixed-point parameters of [6] and compared it to a double-precision floating-point MATLAB reference and the BFO output for a 1 kHz sine at $f_s = 96$ kHz. Our simulations reveal that the method in [6] achieves a total harmonic distortion plus noise (THD+N) of -94.27 dB, which is 42.7 dB worse than what is achieved by our BFO ASIC (see also Table 1). The corresponding power spectra[5] are shown in Figure 7 and it is evident that our CORDIC-based approach generates sine waves with significantly higher purity than the ASIC design reported in [6].

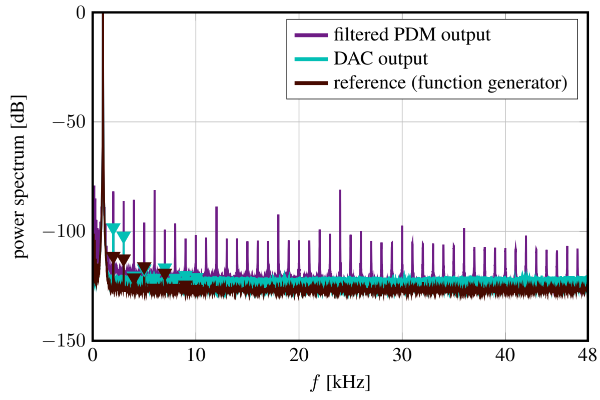
4. MEASUREMENT RESULTS

In order to quantify the performance of the $\pm$synth prototype, we now present a number of measurement results.

[5]We analyze 960 k samples (10 s) using Welch's method with a Hann window, a 2^{14}-point FFT, 50% overlap, and a normalized peak value of 1.



(a) *Power spectra at different stages in the* $\pm$*synth.*



(b) *Power spectra of reference, DAC, and filtered PDM outputs.*

Figure 8: *Measured power spectra (in decibels) for a* 1 *kHz sine wave at a sampling rate of* $f_s = 96$ *kHz.*

4.1. Signal Quality

We first assess the quality of a single partial at different stages of the instrument: (i) the BFO output (I2S output, digital domain), (ii) the DAC output (after reconstruction filter, analog domain), and (iii) the $\pm$synth output (line-level output, analog domain). We generate a 1 kHz sine wave with -6 dBFS amplitude (with respect to the I2S interface) in the BFO at a sampling rate f_s of 96 kHz. The digital BFO output is captured using a logic analyzer to extract raw I2S data; the analog signals are captured using a Focusrite Scarlett 18i20 (3rd gen.) audio interface [16] with a sampling rate of 96 kHz. Figure 8(a) shows the power spectrum of the three signals. The corresponding THD+N values are reported in Table 1. We see that the test signal (sinusoidal) at the BFO output has extremely high purity and the quality is essentially limited by the DAC. We can also see that analog processing through the VCF and VCA circuitry further reduces the THD+N, which is not unexpected.

We also measured the filtered PDM output, which consists of a passive second-order low-pass filter with a -3 dB frequency of 31 kHz. This output achieves a THD+N of -75 dB, which is 13 dB higher than that of the DAC output (see Table 1); the associated spectrum is shown in Figure 8(b). Clearly, the PDM output is inferior to the DAC output, but would enable the use of our BFO ASICs with less expensive external circuitry.

Table 1: *THD+N measurements for a $f = 1$ kHz sine.*

Measurement	THD+N [dB]
$\pm$synth output (analog)	-51.0
PDM output (analog)	-75.0
DAC output (analog)	-88.2
BFO output (digital)	-137.0
reference: function generator (analog)	-89.5

Table 2: *SINAD for different waveforms at $f = 20$ Hz.*

Waveform	SINAD [dB]
sine	134.0
triangle	133.3
sawtooth	135.3
super-saw	131.3
rect-saw	131.0
pulse	109.4

Remark 2. *The audio interface [16] specifies a THD+N below 0.002% (≈ -94 dB) for the line inputs. As a reference, we measured a 1 kHz sine wave generated with an SRS DS360 ultra-low distortion function generator [17], which resulted in a THD+N or -89.5 dB (cf. Table 1); the associated spectrum is shown in Figure 8(b). Since this THD+N result is close to that of the DAC output, our measurements are likely affected by the audio interface.*

Table 2 shows the signal-to-noise and distortion ratio (SINAD) between a MATLAB floating-point model and the digital BFO output for different waveforms generated at a base frequency of $f = 20$ Hz. Most waveforms achieve a SINAD exceeding 131 dB except for the pulse waveform, which yields 109.4 dB. The reason is that since the pulse waveform has the a_k-coefficients of all 1024 partials set to the same value, that value has to be reduced substantially to avoid clipping. Thus, the signal power of this waveform is much lower compared to the other waveforms, which results in lower SINAD.

4.2. Latency

As any digitally controlled instrument, the $\pm$synth exhibits non-negligible latency between a keystroke (or parameter change) and a change in the synthesizer's output. To assess the prototype's latency, we measure the delay between the reception of a note-on MIDI message at the $\pm$synth and the change in signal at the line-level output; this ignores external delays, e.g., caused by the MIDI keyboard. The measurements are repeated 34 times using an oscilloscope probing two signals: (i) the opto-coupler's output in the MIDI receiver circuit and (ii) the hot signal of the line-output. Our measurements show a mean latency of 2.09 ms (minimum 1.60 ms; maximum 2.76 ms; standard deviation 0.33 ms). The latency is mainly caused by the MCU firmware, i.e., processing the UART data (MIDI receiver), computing new control signals, transmitting parameter data over SPI, etc.

4.3. Power Consumption

The $\pm$synth's power consumption is measured for two cases: (i) idle mode (default state after power-up) and (ii) playback (all voices playing). The supply current is measured at 499 mA during idle mode and up to 522 mA during playback. This corresponds to a power consumption of approximately 12 W to 13 W. Since there are many factors influencing the instrument's power (e.g., filter resonance, load at audio outputs, etc.), these measurements should be taken with a grain of salt.

5. LIMITATIONS AND FUTURE WORK

The $\pm$synth prototype, in its current form, has a number of limitations, which we now summarize. First, the BFO ASICs are currently unable to perform oscillator synchronization, which is a direct consequence of the additive-synthesis approach discussed in Section 2.3. To mitigate this limitation, one could load in oscillator parameters $\{a_k, b_k, n_k\}_{k=1}^{K}$ of a synchronized oscillator, but this approach is limited by the rate at which all of the parameters can be rewritten (see Section 3.4). Also, such a workaround might no longer be aliasing-free. Developing hardware-friendly solutions to implement true oscillator synchronization without aliasing, e.g., inspired by the works of [18, 19], is part of ongoing work. Second, the BFO ASICs do not provide hardware support for true additive synthesis with separate envelopes per partial. Such functionality could readily be implemented in hardware, but comes at the cost of additional memory and logic to store and update the envelope parameters. A potential compromise would be to use the linear updates put forward in [6]. Third, the BFO does not provide a noise generator. We are planning to include such missing functionality in a future version. Fourth, the quality of the BFO is currently limited by the used 24-bit DAC, and our audio interface may affect our measurements; in the future, we will use a better DAC to fully exploit the BFO's high purity and also use better measurement equipment. Fifth, we are currently using an off-the-shelf external keyboard and controller—developing a dedicated user interface for the $\pm$synth would be quite exciting.

6. CONCLUSIONS

We have shown the implementation details of the $\pm$synth, an eight-voice hybrid digital-analog music synthesizer prototype that uses aliasing-free digital oscillators followed by analog filtering and amplification. By implementing the oscillators on custom ASICs, we are able to generate a wide variety of waveforms with up to 1024 freely programmable partials per oscillator, which includes not only classical waveforms of analog synthesizers, but also wavetable sounds or bell-like timbres. The $\pm$synth is able to generate a total number of 32 768 partials at a sampling rate of 96 kHz, and measurement results have demonstrated high fidelity and low latency.

While a range of commercial hybrid digital-analog synthesizers became available recently, virtually nothing is known about their inner workings. In contrast, every implementation detail of our hardware prototype is known and well-documented, and every aspect of the instrument can be modified easily. We therefore believe that the $\pm$synth will be an excellent research platform for future real-time audio synthesis experiments.

7. REFERENCES

[1] Timothy Stilson and Julius Smith, "Alias-free digital synthesis of classic analog waveforms," in *Proceedings of the International Computer Music Conference (ICMC)*, August 1996, pp. 332–335.

[2] Victor S. Reinhardt, "Direct digital synthesizers," in *Proceedings of the Annual Precise Time and Time Interval Systems and Applications Meeting*, December 1985, pp. 345–374.

[3] Thomas Schanze, "Sinc interpolation of discrete periodic signals," *IEEE Transactions on Signal Processing*, vol. 43, no. 6, pp. 1502–1503, June 1995.

[4] James A. Moorer, "Signal processing aspects of computer music: A survey," *Proceedings of the IEEE*, vol. 65, no. 8, pp. 1108–1137, August 1977.

[5] Apple Inc., "Alchemy additive element controls in Logic Pro," *Online*, available at https://support.apple.com/guide/logicpro/additive-element-controls-lgsi55ccfb06/10.7.5/mac/12.3, accessed: April 6th, 2023.

[6] Fernando De Bernardinis, Roberto Roncella, Roberto Saletti, Pierangelo Terreni, and Graziano Bertini, "An efficient VLSI architecture for real-time additive synthesis of musical signals," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 1, pp. 105–110, March 1999.

[7] Arturia, "Arturia - Freak," *Online*, available at https://www.arturia.com/ranges/freak, accessed: April 6th, 2023.

[8] Sequential LLC, "Prophet X," *Online*, available at https://www.sequential.com/product/prophet-x, accessed: April 6th, 2023.

[9] UDO Audio, "The super 6," *Online*, available at https://www.udo-audio.com, accessed: April 6th, 2023.

[10] Waldorf Music, "Quantum," *Online*, available at https://waldorfmusic.com/quantum-en, accessed: April 6th, 2023.

[11] Cirrus Logic, *192-kHz Stereo DAC with Integrated PLL*, August 2018.

[12] Sound Semiconductor Inc., *SSI2144: FATKEYS™ Four-Pole Voltage Controlled Filter*, January 2018, Rev. 3.0.

[13] Sound Semiconductor Inc., *SSI2162: FATKEYS™ Dual Voltage Controlled Amplifier*, June 2020, Rev. 1.1.

[14] Jack E. Volder, "The CORDIC trigonometric computing technique," vol. EC-8, no. 3, pp. 330–334, September 1959.

[15] Behrooz Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*, Oxford University Press, Inc., USA, 1999.

[16] Focusrite, "Scarlett 18i20, 3rd generation," *Online*, available at https://focusrite.com/en/usb-audio-interface/scarlett/scarlett-18i20, section: "Specifications in Detail," accessed: April 6th, 2023.

[17] Stanford Research Systems, "DS360 — ultra-low distortion function generator," *Online*, available at https://www.thinksrs.com/downloads/pdfs/catalog/DS360c.pdf, accessed: April 6th, 2023.

[18] Eli Brandt, "Hard sync without aliasing," in *Proceedings of the International Computer Music Conference (ICMC)*, September 2001.

[19] Pier Paolo La Pastina and Stefano D'Angelo, "A general antialising method for sine hard sync," in *Proceedings of the International Conference on Digital Audio Effects (DAFx)*, September 2022, pp. 109–114.