This is an account of a journey into "Binary Sound Synthesis" (BSS), which started early 2005 as a test application[1] for a Forth compiler for the 8–bit Microchip PIC architectures. With BSS is meant the process of generating audible sound from digital square–wave signals using an absolute minimum of logic or code.

This is old stuff. The digital approach dates from the era of early 8–bit game machines, and as such a lot of the algorithms are no longer used. Currently there are few reasons to not use floating or fixed point math with PCM outputs.

The point of this paper is partly to archive and document old techniques, and to shine an idiosyncratic light on the matter. I find this a facinating subject. Probably because it is so different from the standard real and complex number based signal processing.

In BSS the main focus is on the time component, since it is the timing between switching a speaker on and off which becomes the only means of controlling the sound.

Cycles

The simplest sounds we can produce are based on periodic square waveforms. An oscillator can be can be implemented using a frequency divider or pulse counter, which generates one output event for each k input events. Multiple such oscillators can then be connected to a hardware timer interrupt.

Mathematically, this can be related to addition in the ring of integers modulo k. We'll denote this algebraic structure as $\mathbb{Z}_k$, the cyclic group of order k. This section talks about such groups, as they will appear as multiplicative groups of Galois Fields.

[1] The original synthesizer ran early 2005 on a PIC12F628, an 8 bit microcontroller with 6 i/o pins which I had configured to run at 1 MIPS. The current one runs since mid 2006 on a PIC18F1220, an 8 bit micro with 18 pins and a slightly more powerful ISA. The original synth is a synchronous one running at 8kHz. At each sampling point the output state of 3 oscillators is determined. The control updates run at 200Hz, while the note updates run at 8Hz, which is $1/4$ note at 120bpm. The current one uses 3 asynchronous hardware timers, thus a higher maximal rate of 2MHz. In a nutshell, the synth contains 2 main algorithms: cross modulation (XOR mixer), a formant synth (AND/OR mixer), LFSR pseudo-random sequence generator. It uses 3 oscillators that can be synchronized. Most of the interesting parts are in the controller that reconfigures the oscillators.

Counters and Cyclic Groups

Cyclic groups are groups, necessarily abelian, that can be generated by a single element. In our canonical representation, this element will be 1 in $\mathbb{Z}_k$. If $k+1$ is a power of a prime, it is the multiplicative group of the Galois Field GF$(k+1)$, which we will encounter later. All abelian groups are composed of direct products of prime order cyclic groups, but not all abelian groups are cyclic. The prototypical example being the Klein 4–group $\mathbb{Z}_2 \times \mathbb{Z}_2$.

The exact relation between a group and a counter is as follows. We use additive notiation because all the groups are abelian. To each counter we associate a group $\mathcal{G}$, a generator $g \in \mathcal{G}$, and a state $s \in \mathcal{G}$. On each input event, the state s will be replaced with $s+g$. Whenever the state reaches $s=0$, the unit element of the group, an output event is generated. The division factor is the order of g, which is the smallest integer o for which $\underbrace{g+g+\ldots+g}_{o \text{ times}} = 0$.

For example in $\mathbb{Z}_6$ where $\{0, 6, 12, \ldots\}$ denote the same element, the generator 1 has order 6, the generator 2 has order 3 and the generator 3 has order 2.

Composite Numbers

Groups with prime order are not so interesting to us, they generate only one kind of cycle. What interests us most is the combination of timers. Let's have a look at the example of highly degenerate groups, which have a number of elements equal to a composite number like the number of seconds in a week $604800 = 2^7 3^3 5^2 7$.

Note that from the size of an arbitrary group we can't necessarily deduce its structure. However, requiring that the group is cyclic is enough, since there's only one of a given size. This makes cyclic groups look a lot like positive integers. A cyclic group can be constructed explicitly as a product of cyclic groups of prime power order, ensuring the component groups have orders that are mutually coprime. This is again analogous to how positive integers behave. A point of difference, however, is that groups of prime power order are not a product of smaller cycles, but they do contain all smaller prime power cyclic groups as subgroups.

The number above gives the group

$$\mathbb{Z}_{2^7} \times \mathbb{Z}_{3^3} \times \mathbb{Z}_{5^2} \times \mathbb{Z}_7,$$

which consists of 4–tuples with elements from the respective groups. This group is essentially the same as $\mathbb{Z}_{2^7 3^3 5^2 7}$. I'll denote the group order as

$$\#\mathcal{G} = \prod_{n=1}^{N} p_n^{m_n},$$

and $\mathcal{G}$ as the abstract group, where $\mathbb{Z}_{\#\mathcal{G}}$ and $\times_{n=1}^{N} \mathbb{Z}_{p_n^{m_n}}$ are two isomorphic representations. In our representation of $\mathcal{G}$, the cyclic subgroup of order q_n in $\mathcal{G}$ can be generated from the element

$$g_n = \frac{\#\mathcal{G}}{p_n^{m_n}},$$

giving a homomorphism from $\mathcal{G}$ to $\mathbb{Z}_{p_n^{m_n}}$. Combining N such homomorphisms allows the construction of an isomorphism

$$f : \times_n Z_{p_n^{m_n}} \to Z_{\#G} : (x_1, \ldots, x_N) \mapsto \sum_n x_n g_n.$$

This leads us to the more concrete observation that an oscillator based on $\mathcal{G}$ can be implemented either as a single counter $\mathbb{Z}_{\#G}$, using a single generator, or a "wired or" of N counters $\mathbb{Z}_{p_n^{m_n}}$, each with its own generator, where the whole acts as a single frequency divider which only generates an event if all counters have $s_n = 0$.

Musical Scale

This makes one wonder if it's not possible to construct a group that is very degenerate, in a way that it produces a relatively well spread out "frequency spectrum" which maps well to the (logarithmic) frequency resolution of human hearing. Keeping the number of large prime factors small, it might even be possible to create some kind of musical scale with inherent "smoothness". Probably, if there are enough small primes, large prime numbers are not really necessary since frequency resolution for large numbers is less of a problem.

For example $d = 604800 = 2^7 3^3 5^2 7$, which is the number of seconds in a week. Moreover, $d + 1$ is prime, so it has an associated Galois field, but this is no longer a polynomial field,

In order to investigate the usefulness of this, we could plot out the possible frequencies that can be generated using this scheme on a logarithmic scale. On the other hand, using fields might be overkill. Using just cyclic groups (counters) might be better here.

Given a a degenerate cyclic group $\mathcal{G}$, how many different (cyclic) subgroups does it have? All subgroups of cyclic groups are cyclic, so there is a one to one correspondence of the order of all possible combinations of subgroups of $\mathcal{G}$ and the combinations of prime factors of $\#\mathcal{G}$. Which leads us to the number of possible groups

$$\prod_N (m_n + 1),$$

since the subgroups of $\mathbb{Z}_{p^m}$ are $\{\mathbb{Z}_1, \mathbb{Z}_p, \ldots, \mathbb{Z}_{p^m}\}$, which totals $m + 1$.

The subgroups, or numbers representing the orders, can be arranged in a N dimensional cuboid, addressed with coordinates $(x_1, \ldots, x_N)$ where each coordinate is limited to $0 \leq x_n \leq m_n$. The order can be computed as

$$\#\mathcal{G}(X) = \#\mathcal{G}(x_1, \ldots, x_N) = \prod^N p_n^{x_n},$$

which corresponds to the the subgroups $\mathcal{G}(X) = \times^N \mathbb{Z}_{p_n^{x_n}}$ of $\mathcal{G}$

The things we are interested in is the distribution of $\mathcal{G}(X)^{-1}$, since it represents the number of frequences we can generate by dividing a master clock, which would be the CPU clock for example. All frequencies that can be generated in this way have a fairly simple harmonic relationship, making it possible to generate smooth scales and chords.

Playing with Subgroups of $\mathbb{Z}_{p^m}$

Given a divider $\mathbb{Z}_{2^m}$, obtaining one for $\mathbb{Z}_{2^{m'}}$ with $m' < m$ is really straightforward since the event $s = 0$ can be computed modulo $2^{m'}$. This can be implemented as checking for the zero condition after performing an OR operation.

What about the case $p \neq 2$ where the modulo operation is no longer trivial? What happens if we just apply the OR operation anyway? For

example, the 7 element cycle can give the following sequences modulo 2^n.

$$
\begin{aligned}
2^3 &\to 0, 1, 2, 3, 4, 5, 6 \\
2^2 &\to 0, 1, 2, 3, 0, 1, 2 \\
2^1 &\to 0, 1, 0, 1, 0, 1, 0 \\
2^0 &\to 0, 0, 0, 0, 0, 0, 0
\end{aligned}
$$

which gives frequencies $1/7$, $2/7$, $5/7$ and $7/7$. Instead of taking modulo and comparing to zero to generate events, it is possible to access state bits directly and obtain a waveform which changes at the moment of the events discribed above.

The hack I'm after here is to see wether it makes sense to have a couple of static oscillators running and to make noises by directly accessing some of their state bits. The configurable parts are then just the generators used to update the counters. For example, one counter with period $256 = 2^8$, one with $255 = 17^1 5^1 3^1$, one with $225 = 3^2 5^2$, one with $245 = 5^1 7^2$, $250 = 2^1 5^3$ etc....

Binary Signals

This section deals with the analysis tools available to talk about binary signals, and some algorithms to actually generate certain classes of sounds.

Generating Functions

There's not much freedom to combine the output of binary oscillators. The most well–behaved logic operation is XOR, since it preserves all timing information (all transitions). AND and OR act as conditional off/on gates respectively. It is ok to leave out OR, since it is equivalent to AND for inverted signals. The resulting operations are addition (XOR) and multiplication (AND) in the field of integers modulo 2, also known as GF(2). Using this as the base field for analysis of bit sequences s_k allows the use of generating functions

$$s(x) = \sum_k s_k x^k.$$

It's usually more convenient to use single sided sequences, meaning the sum runs over $k \geq 0$, or $s_k = 0$ for $k < 0$. Let's have a look at some oscillators. A pulse train with perioid P is given by

$$s_P(x) = \sum_{k \geq 0} x^{Pk} = \frac{1}{1 + x^P}.$$

Any finite signal represented by the polynomial $s_0(x)$ of degree $< P$ gives rise to a periodic signal by convolving it with the pulse train above or

$$s(x) = \frac{s_0(x)}{1 + x^P}.$$

The polynomial $s_0(x)$ here acts as a finite response filter for the signal $s_P(x)$. A finite pulse of *degree* D, with which I mean $D + 1$ consecutive ones, is represented by

$$s_D(x) = 1 + x + \ldots + x^D = \frac{1 + x^{D-1}}{1 + x}.$$

Difference equations

It is possible to use exactly the same tools for dealing with binary sequences as is customary to do with sequences of real or complex numbers, which means using addition and multiplication of polynomials and power series to represent addition and convolution of sequences. More specifically, solutions to linear difference equations can be expressed as rational functions over GF(2).

For example, the difference equation

$$D^2 y = Dy + y + u,$$

expressed in terms of the generating functions becomes

$$(x^2 + x + 1)y(x) = u(x),$$

which expresses $y(x)$ as the signal $u(x)$ filtered by

$$\frac{1}{x^2 + x + 1} = \frac{x + 1}{x^3 + 1} = (x + 1) \sum_n x^{3n},$$

which is $\sum_n h_n x^n$, with h_n periodic with period 3. To find the sequence corresponding to any rational function, compute the partial fraction expansion in terms of the irreducible factors $p_i(x)$ of the denominator, and perform the computation above, which is to find the minimal degree polynomial $s_i(x)$ such that $p_i(x)^{-1} = s_i(x)(x^P + 1)^{-1}$. Then P will be the period of the sequence generated by $p_i(x)^{-1}$. Above $x^2 + x + 1$ is already irreducible.

So, what we see is that the solution of a linear difference equation in GF(2) is determined by an impulse response (filter), which can be written as a sum of periodic sequences, where each such sequence corresponds to an irreducible polynomial $p(x)$. The period P of such a sequence is the order of the element x in the field of polynomials modulo $p(x)$. These sequences play the role of *damped sinusoids* related to difference equations in $\mathbb{R}$.

Exponentials

To get an analogue of the notion of *complex damped exponential* let's have a look at the splitting field of a single irreducible polynomial

$$\begin{aligned} x^2 + x + 1 &= (x + p)(x + q) \\ &= x^2 + (p + q)x + pq. \end{aligned}$$

The extra field elements p and q satisfy $p + q = 1$ and $pq = 1$. Eliminating p from these equations we obtain $(1 + q)q = 1$ or $q^2 + q + 1 = 0$, which means the terms $0, 1, q, q + 1$ themselves are polynomials in q modulo $q^2 + q + 1$. These elements form the splitting field of the polynomial $x^2 + x + 1$, the quotient field of the polynomial ring GF$(2)[x]$ and its maximal ideal generated by $x^2 + x + 1$. To see that a periodic signal is indeed a sum of "complex" exponentials, we compute a partial fraction decomposition as

an example.

$$\begin{aligned}\frac{1}{x^2+x+1} &= \frac{1}{(x+q)(x+q^{-1})} \\ &= \frac{1}{(1+q^{-1}x)(1+qx)} \\ &= \frac{a}{1+q^{-1}x} + \frac{b}{1+qx} \\ &= \sum_k aq^{-k}x^k + \sum_k bq^k x^k\end{aligned}$$

which is indeed a sum of two exponential sequences $a_k = aq^{-k}$ and $b_k = bq^k$ with values in the splitting field. The values q^{-1} and q are the "signal poles" and the field elements a and b are the analogue of amplitudes or phases. The field has multiplicative order 3 which means both exponentials are of this period, as is their sum. This periodicity can be seen directly by

$$\begin{aligned}\frac{1}{x^2+x+1} &= \frac{1+x}{1+x^3} \\ &= (1+x)\sum_k x^{3k}\end{aligned}$$

Degeneracy

What happens if we have periodicities that cannot be represented as subcycles of the multiplicative group of any extension of GF(2)? All multiplicative groups of the extensions of GF(2) are odd, so can never have a period 2 subcycle.

An example of such a sequence would be a period 2 sequence associated with the generating function $b(x)(1+x^2)^{-1} = b(x)\sum_k x^{2k}$. The factor $b(x)$ doesn't matter much for our point, so we'll choose it such that the partial fraction expansion takes a simple form. The denominator can be factored as $(x+1)^2$, which gives the expansion

$$\frac{1}{x+1} + \frac{1}{(x+1)^2}$$

if we choose $b(x) = x$. The first term is simpy the period 1 sequence $\sum_k x^k$. What about the second one? It is the product of two such sequences, which

can be worked out as

$$\frac{1}{(1+x)^2} = \frac{1}{1+x^2} = 1 + x^2 + x^4 + \ldots = \sum_k x^{2k},$$

wich is a period 2 sequence. It can't be an exponential of an extension field element, so what is it? It turns out it can be expressed as

$$\sum_k (k+1)x^k.$$

This case is of course analogous to $\mathbb{C}$ in the case of a double pole. One tends to forget about these, since in practical signal processing applications they do not occur. This is because the chance a random sequence drawn from a continuous probability distribution over $\mathbb{C}$ has a multple pole is zero. However, GF(2) is finite, so poles with multiplicities greater than one are quite common. There is less room to avoid them!

Any period 2 function has 2 degrees of freedom, so these two terms form a basis for those signals. In general, a period N signal can be reduced to sum of N base terms in the splitting field of $x^N + 1$. We could think of this as a Fourier basis.

A bit more about symbols like $(k+1)x^k$. Here the k actually is a function of x and k which is defined as

$$(k+1)x^k = \underbrace{x^k + x^k + x^k \ldots + x^k}_{k+1 \text{ terms}}.$$

In a similar way we can define the expression

$$\frac{k(k+1)}{2}x^k = \underbrace{1x^k + 2x^k + \ldots + (k+1)x^k}_{k+1 \text{ terms}}.$$

The last one can also be understood as

$$\underbrace{x^k + 0 + x^k + 0 + x^k + \ldots}_{k+1 \text{ terms, including } 0 \text{ terms}}$$

This clearly shows how periods 2^m can be constructed.

More generally, the terms in the partial fraction expansion for a multiple pole p in the splitting field are most conveniently expressed as

$$(1+px)^{-n} = \sum_k \binom{-n}{k} (px)^k$$

If the total period N period is a multiple of a power of 2, or $N = 2^m M$, all poles will have a multiplicity wich is a multiple of 2^m, since

$$x^N + 1 = x^{M2^m} + 1 = (x^M + 1)^{2^m}$$

.

For a period 2^n signal the only pole is $p = 1$. In this case the bit patterns of the base functions take a particular simple form, which consists of a rotated, periodic Sierpinski triangle.

Roots of Unity

There's another way to tell the story — one which does enable any periodic signal to be expressed as a sum of exponentials. However, we need to focus on a different algebraic structure. Using the approach above, the space in which to express the exponentials is a ring constructed as a product of splitting fields of irreducible components, combined with "doubling up" due to multiple poles. This is rather convoluted. It might be easier to directly focus on the structure generated by the roots of unity.

Suppose we can introduce the number ω so it's possible to factor

$$1 + x^N = \prod_k (1 + \omega^k x).$$

Here ω is a generator of the group $\mathbb{Z}_N$, which we have encountered before. The group contains all the cycles with period p_n^k with $k \leq m_n$, since it can be seen as a product of the cyclic groups $\mathbb{Z}_{p_n^{m_n}}$. Here we assume the prime factorization

$$N = \prod_n p_n^{m_n} = 2^{m_1} 3^{m_2} 5^{m_3} \ldots$$

Given any signal with period N, represented by the degree $N-1$ polynomial $s(x)$, the partial fraction expansion in terms of roots of unity is

$$\begin{aligned}\frac{\sum_k s_k x^k}{1+x^N} &= \sum_n \frac{S_n}{1+\omega^{-n}x} \\ &= \frac{\sum_n S_n \sum_k (\omega^{nk})x^k}{1+x^N}\end{aligned}$$

which, after identifying coefficients of x^k gives the relation

$$s_k = \sum_n S_n \omega^{nk},$$

clearly identified as the *discrete Fourier transform* or DFT, in abstract form. To have a look at this transform, we move away from the field of rational functions over GF(2) which we used to express difference equations, and consider only the ring GF$(2)[x]/(1+x^N)$ of polynomials modulo $1+x^N$ as a vector space.

The relation above is an automorphism of the vector space GF$(2)^N$, relating its two representations we shall call x–space and ω–space. It is also a ring isomorphism since it dually maps the operations of multiplication and convolution, to which I'll return later.

But how can we see ω more concretely? In what space do ω and the S_n live? Clearly it does not live in GF(2), or in any extension field of GF(2) in the more general case where N is even. Let's have a look at the example $N = 6 = 2^1 3^1$. In this case it is possible to represent the group generated by ω by an element of GL$(4, \text{GF}(2))$, the general linear group of the 4–dimensional vector space over the field GF(2). If we choose

$$\omega = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

it serves the purpose as a primitive root of unity, since ω^6 is the identity, but lesser powers are all different. The top 2×2 diagonal block of ω is a generator of $\mathbb{Z}_3$, and the bottom one is a generator of $\mathbb{Z}_2$, making this a generator of the cyclic group $\mathbb{Z}_6$ written as an outer product of the former. Writing all linear combinations of the 6 independent elements ω^n gives $2^6 = 64$ elements in the 6–dimensional vector space over GF(2). This vector space is

equipped with a product, turning it into a ring. This ring is isomorphic to the ring of polynomials mentioned above, since the map $x \mapsto \omega$ is clearly an isomorphism, but we already had a different isomorphism, namely the DFT. In general, it is always possible to construct a matrix representation like this.

Now, what is the relation between this way of analyzing periodic signals, and the one explained above?

$\mathbf{GF}(2)[x]/(1+x^N)$

What about this field of polynomials? The extension fields mentioned before can be reconstructed by dividing out some maximal ideals.

Duality

The automorphism of $\mathrm{GF}(2)[x]/(1+x^N)$ we called DFT above is a particular one. We represented this map as a relation between x–space and ω–space. It is easy to see that these spaces are dual with respect to the operations of *componentwise multiplication* and *convolution*.

The exponential representation is exactly what's necessary to analyze componentwise multiplication of sequences, which can be seen here as masking or gating one signal using another, implemented by bitwize AND.

Given two periodic sequences a_k and b_k with periods dividing N, the componentwize multiplication is expressed as

$$\begin{aligned} c_k &= a_k b_k \\ &= (\sum_i A_i \omega^{ki})(\sum_j B_j \omega^{kj}) \\ &= \sum_l (\omega x)^l \sum_{i+j=l} (A_i B_j) \\ &= \sum_l C_l \omega^{kl} \end{aligned}$$

where the sum $i + j = l$ is taken modulo N. It is clear that the coefficients C_l are obtained as a circular convolution of A_i and B_j.

Frequency analysis

It's nice to be able to use rational functions over GF(2) to reason about binary signals, but they don't say much about what we actually hear. We definitely don't hear bit sequences. Much of our hearing is based on the cochlear resonance, and a lot of the subtleties of the binary sequencies we can produce will get lost if the machinery in our inner ear can't make sense of it. That's exactly what we use to our advantage to generate "noise" using periodic bit sequences, which seem to be to GF(2) what sinusoids are to $\mathbb{R}$.

Some good old Fourier series in $\mathbb{C}$ might help us on the road to some interesting patterns. Let's compute the Fourier series of a periodic bit sequence $b_k \in \{0, 1\}$ with period N, here represented as a function $b(t) = \sum_{k=0}^{N-1} b_k(t)$, with $b_k(t) = b_k e(t-k)$, the unit step impulse $e(t) = u(t) - u(t-1)$, where $u(t)$ is the Heaviside step function. This gives the Fourier coefficients

$$\begin{aligned} h_n &= \int_0^N b(t) e^{-\frac{i2\pi nt}{N}} dt \\ &= \sum_{k=0}^{N-1} b_k \int_k^{k+1} e^{-\frac{i2\pi nt}{N}} dt \\ &= \frac{iN(e^{-\frac{i2\pi n}{N}} - 1)}{2\pi n} \sum_{k=0}^{N-1} b_k e^{-\frac{i2\pi nk}{N}} \\ &= \frac{iN(\omega^n - 1)}{2\pi n} \sum_{k=0}^{N-1} b_k w^{nk} \end{aligned}$$

where we take $\omega = e^{-\frac{i2\pi}{N}}$, and of course $h_0 = \sum_k b_k$. This is not the DFT of b_n, because the base functions $b_k(t)$ are not Dirac impulses, and n rangles over the whole numbers. However, we can compute N terms of this infinite sequence from the DFT if we multiply it by the weighting function or spectral envelope $w_n = \frac{iN(\omega^n - 1)}{2\pi n}$. Taking the even and odd part gives scaling factors proportional to $\frac{1}{n}\cos\frac{2\pi n}{N}$ and $\frac{1}{n}\sin\frac{2\pi n}{N}$.

Now, the question is, how to exploit possible means for generating bit sequences b_k to get to a desired or otherwize interesting h_n?

Linear Transforms

We already encountered linear transforms of vector spaces over GF(2) to express the Nth roots of unity, yielding a representation of the ring of polynomials modulo $1 + x^N$. Some more about representations.

An example. The algebra of linear transforms $\mathrm{GF}(2)^{2\times 2}$ contains 16 elements. Any element A in this algebra can be related to its characteristic polynomial

$$a(x) = \begin{vmatrix} a_{11} - x & a_{12} \\ a_{21} & a_{22} - x \end{vmatrix},$$

which has the general form

$$a(x) = a_2 x^2 + a_1 x + a_0.$$

Any element A satisfies its own characteristic polynomial or $a(A) = 0$. The result of this is that the ring $\mathrm{GF}(2)/(a(x))$ can be embedded in the matrix algebra.

For $\mathrm{GF}(2)^{2\times 2}$ there are two interesting subrings, apart from the trivial ring and GF(2). The first one comes from $x^2 + x + 1$, which is the characteristic polynomial of

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix},$$

giving the field $\mathrm{GF}(2^2)$, with elements represented as the matrices $\{0, I, A, A^2 = A+I\}$. This field has the cyclic group $\mathbb{Z}_3$ as a multiplicative group, consisting of $\{I, A, A^2\}$. The second one comes from $x^2 + 1$, which is associated to the matrix

$$B = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix},$$

giving the ring $\mathbb{Z}_4$ with elements $\{0, B, B^2 = I, B + I\}$. This ring has the cyclic group $\mathbb{Z}_2$ as a group of units, consisting of $\{I, B\}$, and a zero divisor $(B + I)^2 = 0$. The isomorphism from this representation to the integers modulo 4 is obviously $\{(0, 0), (I, 1), (B, 3), (B + I, 2)\}$. The matrix

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

gives the same structure, since it has the same characteristic polynomial.

Sierpinski Triangle

Let's have a look at some recursive combinations of the matrices above. For example, call $B_0 = [1]$ and

$$B_{n+1} = \begin{bmatrix} B_n & B_n \\ 0 & B_n \end{bmatrix}.$$

Squaring this gives

$$B_{n+1}^2 = \begin{bmatrix} B_n^2 & 0 \\ 0 & B_n^2 \end{bmatrix} = I_{n+1}$$

which doesn't bring us very far apart from saying that the upper triangular arrangement of the Sierpinski triangle gives involutions. We could have gotten here too by recursively constructing the antidiagonal from the other matrix which generates $\mathbb{Z}_2$.

$$A_{n+1} = \begin{bmatrix} A_n & A_n \\ A_n & 0 \end{bmatrix},$$

where $A_1 = A$? This is the symmetric arrangement of the triangle. Squaring this gives

$$A_{n+1}^2 = \begin{bmatrix} 0 & A_n^2 \\ A_n^2 & A_n^2 \end{bmatrix},$$

and the third power gives

$$A_{n+1}^3 = \begin{bmatrix} A_n^3 & 0 \\ 0 & A_n^3 \end{bmatrix} = I_{n+1},$$

which gives a similar relation to the one for the iterated tensor product of B.

Walsh Functions

Over $\mathbb{R}$, Hadamard matrices are defined as the successive tensor products of

$$H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix},$$

where $H_1 = \left[\ 1\ \right]$ and

$$H_{2^{n+1}} = H_2 \otimes H_{2^n} = \left[\begin{array}{cc} H_{2^n} & H_{2^n} \\ H_{2^n} & -H_{2^n} \end{array}\right].$$

Walsh matrices are obtained from these by arranging the rows so that the number of sign changes is in increasing order. This is called *sequency* ordering. This permutation is equal to bit reversal permutation, followed by Gray coding.

These matrices have the property $H_n H_n^T = nI_n$. This relation also holds for matrices with integer coefficients. However, mapping this directly to GF(2) renders the relation trivial, since the matrices have all 1 elements and become nilpotent. How can these be related to 2^nth roots of unity represented as matrices over GF(2)?

One obvious embedding of roots of unity is using the circular shift operator. Are there any with a more interesting pattern, but still with characteristic polynomial $x^N + 1$?

Oddities of GF(2)

In GF$(2)[x]$, every irreducible polynomial has to have a nonzero coefficient of x^0, so $p(x) + 1 = xq(x)$. Then $q(x) = x^{-1} \mod p(x)$. This gives easy access to the operations (approximating) left and right shift of the bit vector of coefficients, which are represented as multiplication by x and x^{-1} respectively.

The polynomial $x + 1$ generates the binomial triangle. In GF$(2)[x]$ this is the higly symmetrical Sierpinski triangle. As a consequence we have relations like $(x^a + x^b)^2 = x^{2a} + x^{2b}$.

Application : Linear Noise Generators

The cheapest way to generate noise for the application at hand is to use exactly the impulse response of linear systems described by difference equations as explained above. If the period of the bit sequence is large enough, the human auditory pattern detection will not be able to recover the redundancy.

Following the same algebraic trick as exposed above, suppose the difference equation is expressed by $p(x)y(x) = u(x)$, and $u(x) = 1$, corresponding to the impulse sequence $(1000\ldots)$, what we need to do is to find

out which period is related to $p(x)$. In other words, we have to find the $s_p(x)$ with the lowest possible degree such that

$$\frac{1}{p(x)} = \frac{s_p(x)}{x^P + 1} = s_p(x) \sum_n x^{Pn},$$

where P is the period of the bit sequence corresponding to $p(x)^{-1}$. This is equivalent to the congruence relation

$$x^P = 1 \mod p(x).$$

To get the smallest P satisfying this equation as large as possible, it is necessary to choose $p(x)$ to be a *primitive* polynomial, which means it is irreducible over GF$(2)[x]$, and $x^n \mod p(x)$ generates all nonzero elements in the field modulo $p(x)$, also denoted as GF(2^n) with n the degree of p.

The direct implementation of the difference equation is called a *Linear feedback shift register* or LFSR. The next output is computed as the sum of a certain number of taps. I originally used a 16bit LFSR with taps $(15, 14, 12, 3)$, but the parallel variant is simpler to implement given a parallel XOR operation is available.

This parallel variant is related more directly to the Galois field GF(2^n). More specificly, the output of the generator is taken to be any coefficient of the polynomial sequence generated by x, or $\sum_k s_{k,n} x^k = s(x)_n = x^n \mod p(x)$, where the output sequence could be $s_{0,n}$.

To illustrate the difference between an irreducible and a primitive polynomial, take the irreducible polynomials $x^8 + x^4 + x^3 + x^2 + 1$ and $x^8 + x^4 + x^3 + x + 1$. Both are irreducible, but only the former produces a primitive field. In the field of polynomials modulo the latter one, the $x+1$ is a generator, but not x. Note that these fields are isomorphic, or essentially equal, but their representations are different.

Implementing the parallel scheme for the former polynomial on a digital computer is straightforward. Multiplication of the state polynomial $s(x)$ by x corresponds to a left shift of the register containing the coefficient bits. The bit that's shifted off (the coefficient of x^8) is replaced by adding the other terms $x^4 + x^3 + x^2 + 1$ to the current state. In other words, when a bit carries over, `1Dh` $=$ `00011101b` is XORed with the state.

Other Discrete Structures

Phase and Frequency Modulation

What happens when we modulate the generators of an additive subgroup, or period of counters? Let's call the first *phase* modulation and the second *frequency* modulation.

Using an oscillator with period M and generator 1 to generate the generator of a second oscillator F gives the following function

$$F_n = \sum_{i=0}^{n} i = \frac{n(n-1)}{2} \mod F.$$

Cellular Automata

Another interesting way to generate signals is by using cellular automata. Using the ideas above, we could represent a finite row of nonzero cells as a polynomial, and the update function as another polynomial. Giving rise to an update function in the ring GF$(2)[x]$ or any finite field GF$(2)[x]/(p(x))$.

The NNT and the Fields GF$(2^{2^n}+1)$

For $0 \leq k \leq 4$ the Fermat numbers $F_k = 2^{2^k}+1$ are prime, which means there exists a primitive field $\mathcal{F}_k = \text{GF}(F_k)$. This field is isomorphic to $\mathbb{Z}/F_k\mathbb{Z}$ the integers modulo F_k.

Implementing this representation requires the implementation of a modulo operation. This can be quite expensive, especially on cheap hardware lacking a fast hardware divider. However, it is possible to embed a representation of $\mathcal{F}_k$ in the ring $\mathcal{R}_{k+1}$, where we take $\mathcal{R}_k$ to be the ring represented by $\mathbb{Z}M_k/\mathbb{Z}$, the integers modulo $M_k = 2^{2^k}-1$, where the $\{M_k : k \in \mathbb{N}\}$ form a subset of the set of Mersenne numbers $\{2^n - 1 : n \in \mathbb{N}\}$.

The implementation of this ring relies on the reduction modulo M_k, which is a cheap operation if $K = 2^k$ is (a multiple of) the machine word-size. Suppose we want to reduce a number $[c : a]$ of $2K$ bits modulo M_k,

represented by the concatenation of two K bit words c and a. This gives

$$\begin{aligned}[c:a] &= 2^K c + a \\ &= (2^K - 1)c + c + a \\ &= c + a\end{aligned}$$

with operations modulo M_k. The consequence of this is that both addition and multiplication in $\mathcal{R}_k$ can be performed by the default finite wordlenght unsigned integer operations, followed by this very simple reduction step. This reduction is just addition due to the availability of c as a separately addressable entity, namely carry bit for unsigned addition and high word for unsigned multiplication.

So how to embed $\mathcal{F}_k$ in $\mathcal{R}_{k+1}$? The map

$$f_k : \mathcal{R}_{k+1} \to (2^{2^k} - 1)\mathcal{R}_{k+1} : x \mapsto x 2^{2^k - 1}(2^{2^k} - 1) = x 1_k$$

will do just that. Here

$$(2^{2^k} - 1)\mathcal{R}_{k+1} = \{(2^{2^k} - 1)m : m \in \mathcal{R}_{k+1}\},$$

which we will call I_k is the ideal generated by $2^{2^k} - 1$ in the ring $\mathcal{R}_{k+1}$. The element

$$1_k = 2^{2^k - 1}(2^{2^k} - 1)$$

is the representative in $\mathcal{R}_{k+1}$ of the multiplicative unit of $\mathcal{F}_k$. It can be used to generate the additive group, which gives us a clear relation between integers and the representation in $\mathcal{R}_{k+1}$ of their associated elements in $\mathcal{F}_k$.

So how can we see this is a representation of $\mathcal{F}_k$? It is easy to see that f_k is a group homomorphism for the additive group of $\mathcal{R}_{k+1}$ since $f_k(a+b) = (a+b)1_k = a1_k + b1_k = f_k(a) + f_k(b)$. To see that f_k is a ring homomorphism, rests us to show that f_k is a group homomorphism for the multiplicative group. We need $f_k(a)f_k(b) = ab(1_k)^2 = ab1_k = f_k(ab)$ which is valid if $1_k^2 = 1_k$. That this is so can easily be verified.

The kernel $\{x : f_k(x) = 0\}$ of this ring homomorphism is $\{xF_k : x \in \mathcal{R}_{k+1}\}$, which is a maximal ideal of $\mathcal{R}_{k+1}$, which means that $\mathcal{R}_{k+1}/\{xF_k\}$ is a field. This structure is carried into the ideal $M_k\mathcal{R}_{k+1}$ by f.

To find this representation by construction see `mersenne.tex`, which uses arguments to identify 1_k in the additive subgroup generated by M_k, by requiring the property $1_k^j = 1_k$, for $0 \leq j \leq F_k$,

Shape of 1_k

Returning to the embeddings of $\mathcal{F}_k$ in $\mathcal{R}_{k+1}$, the interesting thing what 1_k exactly looks like.

$1_0 = \square\blacksquare$
$1_1 = \square\blacksquare\blacksquare\square$
$1_2 = \square\blacksquare\blacksquare\blacksquare.\blacksquare\square\square\square$
$1_3 = \square\blacksquare\blacksquare\blacksquare.\blacksquare\blacksquare\blacksquare\blacksquare.\blacksquare\square\square\square.\square\square\square\square$
$1_4 = \square\blacksquare\blacksquare\blacksquare.\blacksquare\blacksquare\blacksquare\blacksquare.\blacksquare\blacksquare\blacksquare\blacksquare.\blacksquare\blacksquare\blacksquare\blacksquare.\blacksquare\square\square\square.\square\square\square\square.\square\square\square\square.\square\square\square\square$

Here the dots just separates nibbles. This looks like a nice square wave! Another interesting thing to see is that in $\mathcal{R}_{k+1}$, there is only one silence, which is all zero, since it is the same element as all one.

What do the other numbers in the ideal $M_k\mathcal{R}_{k+1}$ look like? The redundancy is $M_k : 1$, which gets large quite fast. Are the waveforms special in some way so we can use them as sounds?

Resonant Filter

Using two state output, there is really no straightforward way to add two signals, performing additive synthesis. Composing a signal consisting of different frequencies by adding them together modulo 2 behaves more like cross modulation than addition. The audible spectrum will contain not only the sum, but also the modulation components.

However, it is possible to approximate 2 tone polyphony by synthesizing waveforms that have two clearly distinct harmonic components. The approach is based on the following observation. The sequence

$$\blacksquare\square\blacksquare\square\blacksquare\square\square\square\square\square\square\blacksquare\square\blacksquare\square\blacksquare\square\square\square\square\square\square\ldots$$

resembles the response of a resonant filter with period 2 and time constant 6, to an impulse signal with period 12, while the sequence

$$\blacksquare\square\blacksquare\square\blacksquare\square\blacksquare\blacksquare\blacksquare\blacksquare\blacksquare\blacksquare\square\blacksquare\square\blacksquare\square\blacksquare\square\square\square\square\square\square\ldots$$

resembles a response of the same filter to a square wave with period 24. It is even so that this particular scenario can be expressed in terms of generating functions, when the resonant filter is represented by a polynomial (finite), stretching the analogy even further.

This system has 3 degrees of freedom: the periodicity, the resonance frequency and the time constant. This can be implemented as the product (AND) of a pulse width modulated low frequency signal oscillator, and a high frequency oscillator which has its phase reset for each $0 \rightarrow 1$ transition of the low frequency signal.

It is probably possible to combine two of those to get two formants which is enough to encode very basic vowel spectra. This requires 3 timers, which fits perfectly in the available hardware.

Chaotic Oscillator

An emulation of the qualitative behaviour of a chaotic oscillator can be obtained by driving the Resonant Filter system described above with a base period that is irregular.

Pulse Width Modulation

Now, this is sort of cheating, since the purpose of PWM is really efficient switched digital to analog conversion. Using it, we give up the "true" binary output, and add a layer that will allow us to vary the voltage (on average) between more than 2 levels.

The `18f1220` chips contain a hardware PWM module with 8 bit resolution (10 bit for the enhanced version). A fixed period oscillator is used as the timing source to drive a pin high and low with a programmable duty cycle.

I do wonder wether it's possible or desirable to do some computations in GF(257). I don't see much however, other than convolution using NTT.

$\Sigma\Delta$ Modulation

Note that if switching efficiency isn't all important, $\Sigma\Delta$ modulation gives much better performance for 1–bit audio digital to analog conversion. It generates higher frequency waveforms, but better low frequency resolution.

It can also be used as a cheap way to do synthesis, since it behaves approximately as a reverse frequency to voltage converter: output pulse frequency is on average proportional to the input value.

With multibit digital input, it is very easy to implement: the binary $\Sigma\Delta$ signal is the carry bit of a simple input accumulator. When the accumulator overflows, one "packet" of energy is transferred to the output.

Gray Code

Gray codes are (cyclic) sequences of N binary digits, where each number in the sequence differs only by one bit from the previous. They can be understood as *Hamiltonian paths* on a N–dimensional hypercube, where one moves along edges of the cube.

However, the most commonly used gray code is one with a particular kind of symmetry. It is called the *binary reflected* grey code or BRGC, which is a recursive scheme that traverses both $N-1$ dimensional halves of the hypercube in the opposite direction. This rule is applied recursively and terminates at the trivial 1 dimensional case.

If $\bar{s}$ is the reversal of the sequence s, $s : t$ is the concatenation of sequences s and t, and $b(s)$ prepends each bit vector in the sequence s with bit b, the BRGC sequence recursion relation is

$$s_{n+1} = 0(s_n) : 1(\bar{s}_n),$$

where s_0 is the length 1 sequence containing the empty vector.

Control and Note Sequencing

This is a whole other ballpark. With *control* is meant the evolution of the configuration of the synth engine at a slower rate. This is called *haptic* rate and is related to our ability to experience mechanic vibrations. The next rate down is *note* rate, and roughly corresponds to the frequency of events that can be experienced visually. This cascade of time scales is called the *frequency hierarchy*.

The lower frequency parts are much less limited by the speed and simplicity of the hardware, since more time can be used to do actual computations. This makes it rather pointless to try to exhaust the possibilities. However, here are some general ideas.

Transient Stack

In a binary synth there is no direct way to mix sounds. One way of bringing discrete events to the forground is by temporarily interrupting stable background sounds.

The typical example would be to synthesize a sequence with *bass*, *bassdrum*, *snare* and *hihat*, which are ordered here left to right with time extent decreasing. Mixing the 4 will boil down to playing the shortest sound until it is done.

Wavetable

Since there's a wavetable player, it is possible to do some control rate timbre synthesis by generating or updating wavetables. There are a lot of possibilities here. Maybe the most interesting parts are transform domain methods like the *Walsh–Hadamard* transform.

Chirps

A simple form of frequency modulation using a linear ramp or a sawtooth signal is easily implemented. This can be used to generate drum-like sounds.

Design and Implementation

Frequency Hierarchy

Human perception already seems to have a built in time hierarchy roughly ordered from high to low as *hearing*, *feeling*, *seeing*, *moving*, *remembering*. This is just a vague description, since a lot of these time scales overlap. However, this idea of time hierarchy can easily be reflected in the way we keep time in the synth, using the concepts of *sample*, *control* and *note* frequency.

This is fairly standard practice. We're going a step further in that each of these levels has an associated language, which metaprograms the previous one. This is possible since efficiency can be traded for expressive power as we move up the time scale ladder.

N) Note state updates switch control engines.

C) Control state updates switch synth engines, synth parameters and signal routing.

S) Synth updates produce waveform samples.

It's probably easiest to have the low levels pre–empt the higher ones, so computations can be spread out without explicit cooperative multitasking. There's no reason to not have cooperative multitasking on the high abstraction layer though.

Synth Architectures

- An asynchronous design using the 3 high resolution timers to generate events, and the low resolution timer to drive the noise generator and the other time bases.
- A synchronous design with a software synth engine running at a fixed sample rate, from which all other time scales can be derived.
- A similar synchronous design, but using 10 bit hardware PWM as output.

Synth Patch

The idea is to make the synth as configurable as possible, but at the same time keeping the configurability contained in a small number of bits so the transient stack can be implemented efficiently. The way to do this is to eliminate all redundancy (symmetry) in the configuration data.

Also the decoding of this information should be very straightforward, since it has to be computed at each synchronous or asynchronous engine event. I'll call the total configuration state, not including the engine state, as the *synth patch*.

The synth engine consists of 4 *generators*. One noise generator `[noise]` and 3 square wave oscillators `[osc0]`, `[osc1]` and `[osc2]`. The oscillators and the noise generator each have an associated update period, which is part of the patch data. There state is not.

These oscillators can be synced to each other as single master sync $0 \rightarrow 1, 2$ or cascaded sync $0 \rightarrow 1 \rightarrow 2$. This is encoded as one bit for `[osc1]` and two for `[osc2]`, leaving room for one external sync event.

I'd like to parametrize the output combination of the 3 square wave oscillators and the noise generator using a limited set of boolean functions. For 4 inputs, there are $2^{2^4} = 2^{16}$ such functions. To see this, draw the Karnaugh map of a 4 input function, and observe it has 16 squares in total. Some things that could be divided out are: the polarity of the output, the polarity of the noise signal and the polarity of each of the input square waves.

Working it from the other side, the necessary combinations to get the absolute minumum functionality are

- `[xmod]` Add (XOR) the output of 3 square wave outputs. This can be used to generate PWM.
- `[reso]` Multiply (AND) `[osc1]` and `[osc2]`, and add (AND) with `osc0`.
- `[gate]` Multiply (AND) the output of 3 square wave outputs.

Synth Algorithms

Currently, there's only 3. Square waves, pseudorandom noise and sample playback. Together with intermodulation schemes described above, this gives already quite a rich palette.

Misc Hacks

So evaluate AND of a number of bits in a byte, mask out all the bits that are not necessary, and use the ADD operation. This can also be used to evaluate expressions of the form $f = a + bc$, which occur in the `[reso]` mixer. Here I combine 3 waveforms as $f = a + bc$. Using ordinary adder logic this gives $(a, b, c) + (0, 0, 1) = (f, x, x)$. Adding more bits allows to propagate the result into the carry flag.

Rotating oscillator period's bit rep.

Electronics

I think a lot can be done by combining analog and digital electronics. One of the prototypical examples is of course the analog time varying reso filter. Some other interesting things could be done by using comparator based chaotic oscillators (switched unstable systems) and time constant capturing.