BLAKE TROISE

The 1-Bit Instrument

The Fundamentals of 1-Bit Synthesis, Their Implementational Implications, and Instrumental Possibilities

ABSTRACT The 1-bit sonic environment (perhaps most famously musically employed on the ZX Spectrum) is defined by extreme limitation. Yet, belying these restrictions, there is a surprisingly expressive instrumental versatility. This article explores the theory behind the primary, idiosyncratically 1-bit techniques available to the composer-programmer, those that are essential when designing "instruments" in 1-bit environments. These techniques include pulse width modulation for timbral manipulation and means of generating virtual polyphony in software, such as the pin pulse and pulse interleaving techniques. These methodologies are considered in respect to their compositional implications and instrumental applications. KEYWORDS chiptune, 1-bit, one-bit, ZX Spectrum, pulse pin method, pulse interleaving, timbre, polyphony, history

INTRODUCTION

As unquestionably evident from the chipmusic scene, it is an understatement to say that there is a lot one can do with simple square waves. One-bit music, generally considered a subdivision of chipmusic,[1] takes this one step further: it is the music of a single square wave. The only operation possible in a 1-bit environment is the variation of amplitude over time, where amplitude is quantized to two states: high or low, on or off. As such, it may seem intuitively impossible to achieve traditionally simple musical operations such as polyphony and dynamic control within a 1-bit environment. Despite these restrictions, the unique techniques and auditory tricks of contemporary 1-bit practice exploit the limits of human perception. Through layers of modulation, abstraction, and perspicacious writing, these compositional methods generate music far more complex than the medium might, at first impression, suggest. Even if not originally conceived through ludic platforms (one can hear simple examples of 1-bit sonics in microwave interfaces and smoke alarms) 1-bit music, as it is understood today, has been developed and propagated through video games and the companion demoscene culture.[2] Where systems such as the ZX Spectrum and early desktop computers contained severely limited audio capabilities, developers found creative solutions to do the seemingly impossible: polyphony, timbral variation, and dynamic volume, all using

1. YERZMYEY, "1-Bit Chiptunes / Beeper Music," 2013, http://chipmusic.org/forums/topic/12566/1bit-chiptunes-beeper-music/.

2. Antti Silvast, Markku Reunanen, and Gleb Albert, "Demoscene Research," http://www.kameli.net/demoresearch2/.

Journal of Sound and Music in Games, Vol. 1, Number 1, pp. 44–74. e-ISSN 2578-3432. DOI: https://doi.org/10.1525/jsmg.2020.1.1.44.

a single, monophonic square wave. These tricks (often born through necessity) have established a broad and expressive 1-bit instrumental idiolect to rival any acoustic instrument.

The *1* in 1-bit indicates the number of binary digits (bits) required to describe the signal. Values in (most) computational systems are represented by strings of binary numerals, which are commonly abstracted using the symbols 0 and 1. In reality, this data can be understood by whichever interpretation is most relevant to the paradigm. For example, if we were to represent a low amplitude with a 0 and a high amplitude with a 1, the amplitude could be described by one bit. This would allow no more than two possible amplitudinal positions. If a value were to include multiple bits, larger numbers could be represented; a value consisting of two bits can be in four possible configurations—00, 01, 10, and 11—and, with each bit added, the number of possible configurations increases by powers of two.[3] Although seemingly etymologically identical, 1-bit music is distinct from the terms *8-bit* and *16-bit music* as these terms refer (somewhat liberally) to the architectures of the platforms these musics were historically associated with.[4] Using the same definition as the 1-bit terminology, Nintendo Entertainment System (NES) music might perhaps be better described as 4-bit, as there are sixteen possible amplitudinal states each pulse (and noise) channel can occupy.[5] The 1-bit soundscape is capable of performing many of the chipmusic techniques typical of the NES, Game Boy, and Commodore 64,[6] however, unlike these systems, the limitations are dictated not by audio hardware but by the *software* implementation. This arises because the audio is generated entirely by the software routine, with no delegation to external hardware (like sound chips). One-bit music is typically generated by the CPU alone, often requiring significant amounts of calculation time compared to other methods of synthesizing audio. This is both the medium's greatest weakness, because sound routines may leave too little processing time for other tasks, and greatest strength, as the synthesis can be determined in software alone and dictated by the composer-programmer. I use the term

3. If we were to assign an amplitudinal value to each of these four combinations, there would certainly be more states for the amplitude to occupy, but we would not be able to easily sonify these states. Computers are digital machines, thus their constituent circuitry works using two voltage levels: one voltage to represent the binary 0 and another to represent the 1. To achieve greater amplitudinal resolution, purpose-built external circuitry must be used to convert these values into an analog signal—a signal that may be a continuity of voltage levels. So why is 1-bit music a solution at all? In short, adding more hardware is expensive, whereas attaching a speaker (almost) directly to a CPU output pin and reading binary voltage levels is much cheaper.

4. Joe Allen, "How to Make 8-Bit Music: An Introduction to Famitracker," *Synthtopia*, May 2015. http://www.synthtopia.com/content/2015/05/01/how-to-make-8-bit-music-an-introduction-to-famitracker/; Gerald Lynch, "From 8-Bit to Chiptune: The Music That Changed Gaming Forever," *Techradar*, March 2017, http://www.techradar.com/news/8-bit-music-the-soundtrack-to-a-gaming-revolution-that-resonates-today; Liz Ohanesian, "What, Exactly, Is 8-Bit Music?" *LA Weekly*, August 2011, http://www.laweekly.com/music/what-exactly-is-8-bit-music-2409754.

5. blargg, "NES Apu Sound Hardware Reference," nesdev.com, 2004, http://nesdev.com/apu_ref.txt.

6. For an audio example, I direct the reader to the following ZX Spectrum pieces: raphaelgoulart's "surprisingly NOT four twenty" (ZX Art, 2014, https://zxart.ee/eng/authors/r/raphaelgoulart/surprisingly-not-four-twenty/) and Brink's "M'Lady" (ZX Art, 2014, https://zxart.ee/eng/authors/b/johan-elebrink/mlady/), both of which follow the classic programmable sound generator (PSG) model. In short, this is characterized by instrumental figures such as super-fast arpeggios and the treatment of single oscillators as individual instruments (Christopher Hopkins, "Chiptune Music: An Exploration of Compositional Techniques Found in Sunsoft Games for the Nintendo Entertainment System and Famicom from 1988–1992," PhD diss., Five Towns College, 2015).

composer-programmer to describe the individual who creates not only music but also its software encapsulation, the *sound routine*.[7] In wider computing practice, a routine is a portion of executable code that performs a task of specific function. For example, a software routine might read input keypresses from a keyboard, check a document for spelling errors, or display text on a monitor. A *sound* routine is concerned with the generation of audio, generally with a focus on synthesizing music. The term *routine* is here used synonymously with *software* or *program*, in keeping with demoscene and chipmusic nomenclature.[8]

This is an immensely exciting medium for the musician; the primary input of the composer-programmer in this framework is their creative expression when designing routines for musical playback. The choices made in the design of the routine dictate which musical possibilities are available in subsequent compositions. There are multiple 1-bit routines for the ZX Spectrum that implement identical concepts and synthesis techniques, yet each of these routines develops and expands on different facets, making concessions on one feature for increased focus on another. For example, Tim Follin's early "3ch routine"[9] forgoes accurate pitch replication and quantity of channels for smaller file sizes, whereas Ján Deák's ZX-7 engine implements eight software channels with more accurate tunings, but makes concessions on RAM buffer space.[10]

I intend for this article to serve as an introduction for those who have not yet encountered 1-bit instrumental practice and, perhaps most importantly, as a comprehensive guide for would-be composer-programmers who wish to create their own 1-bit music—irrespective of chosen platform. Although there are a wealth of instrumental possibilities and potential features, fundamentally the first decision the 1-bit programmer must make when designing a routine is a negotiation between polyphony and timbre. Classic chipmusic techniques and 1-bit music have significant overlap, and many of the instrumental solutions are cross-applicable, but 1-bit music is sonically unique in this negotiation between polyphony and timbre.

A SHORT HISTORY OF 1-BIT MUSIC

The history of 1-bit music is inexorably linked to the history of computational music; 1-bit synthesis often presents itself as the simplest solution when generating digital audio (see "Fundamentals" section). The earliest examples of computationally synthesized music emerge with the advent of programmable electronic computers in the post-war era of the

7. See, for example, the work of David Warhol (Patrick "Bucky" Todd and Steve "Apoc" Lakawicz, "Interview with David Warhol (Composer, Programmer)," *Video Game History*, December 2016, http://www.vgarc.org/vgarc-originals/interview-with-david-warhol/), Tim Follin ("Star Tip 2," *Your Sinclair* no. 20 (1987), 201–13), and Dave Wise ("Interview with David Wise (December 2010)," Square Enix Music Online, 2010, https://www.squareenixmusic.com/features/interviews/davidwise.shtml).

8. Kevin Driscoll and Joshua Diaz, "Endless Loop: A Brief History of Chiptunes," *Transformative Works and Cultures* 2 (2009); Philip Phelps, "A Modern Implementation of Chiptune Synthesis," University of the West of England, 2007, https://woolyss.com/chipmusic/chipmusic-discovery/PhillPhelps-ChiptuneSynth.pdf; utz, "Tutorial: How to Write a 1-Bit Music Routine," 1-Bit Forum, July 2015, http://randomflux.info/1bit/viewtopic.php?id=21. Alternatively, *driver* is sometimes used, though the term more commonly refers to event sequencing programs written to interface with outboard PSGs.

9. Tim Follin, "Star Tip 2," *Your Sinclair* no. 20 (1987), 201–13.

10. ZX-7. 1991. Ján Deák / Elektronika . ZX Spectrum.

twentieth century.[11] Consequently, the first recorded instance of a digital composition was a 1949 program written for the BINAC computer by Frances "Betty" Holberton (then Betty Snyder). Jack Copeland and Jason Long's claims and 1-Bit Forum user utz's (independent) research[12] both make a very compelling case for this assertion, which changes the origin of computer music from the often cited CSIRAC and Manchester Computer musical software of 1951 to two years prior.[13] The very first sequenced music for an electronic computer seems to have been Holberton's rendition of "For He's a Jolly Good Fellow" to celebrate the completion of the BINAC project.[14] In the 1950s, Alan Turing outlined the theoretical basis for computational synthesis in the *Programmers' Handbook for Manchester Electronic Computer Mark II*, where an audio routine is proposed.[15] The described application for generating sound is pragmatic rather than creative: Turing's "Hooter" function generates tones, clicks, and pulses as a feedback system for computer-human interaction. The document suggests that the generated tones might allow a computer's operator to "listen in" to the progress of a routine, somewhat similar in function to later technologies such as the dial-up modem. Despite Turing's initial intentions, this routine was eventually employed to create the earliest surviving computer music: a monophonic rendition of "God Save the Queen."[16] Both the BINAC routine and the "Hooter" function would, most likely, have used 1-bit synthesis; the method of generation described by BINAC engineer Herman Lukoff suggests a process similar to contemporary 1-bit routines: ". . . by programming the right number of cycles, a predictable tone could be produced. So BINAC was outfitted with a loudspeaker . . ."[17] Additionally, Turing's "Hooter" employed a series of clicks, which suggests a method similar to the very thin pulse widths used in ZX Spectrum sound routines (see "Timbre" section).[18] There are numerous subsequent examples in the 1950s of music created using research and military computers, the majority offering similar, monophonic

11. Contrastingly, the earliest citation I could find of computational music is much older: a speculative musing by Ada Lovelace in the early nineteenth century—a century ahead of its actualization ("Ada Lovelace," Computer History Museum, http://www.computerhistory.org/babbage/adalovelace/).

12. Jack Copeland and Jason Long, "Christmas Carols from Turing's Computer," *Sound and Vision Blog*, 2017, https://blogs.bl.uk/sound-and-vision/2017/12/christmas-carols-from-turings-computer.html; utz, "Topic: A Timeline of 1-Bit Music (1949-1979)," 1-Bit Forum, 2015, http://randomflux.info/1bit/viewtopic.php?id=40; utz, "Computer Music in 1949?" Ancient Wonderland, irrlicht project, November 2015, http://irrlichtproject.blogspot.co.uk/2015/11/computer-music-in-1949.html.

13. Anders Carlsson, "TIMELINE," ChipFlip, 2017, https://chipflip.wordpress.com/timeline/; Jonathan Fildes, "'Oldest' Computer Music Unveiled," Technology, BBC News, June 2008, http://news.bbc.co.uk/1/hi/technology/7458479.stm; "First Digital Music Made in Manchester," Technology, The University of Manchester, June 2008, http://www.manchester.ac.uk/discover/news/first-digital-music-made-in-manchester.

14. Kathryn Kleiman, "Singing Binac - 1948," CYHIST Community Memory: Discussion List on the History of Cyberspace, November 1997, https://groups.yahoo.com/neo/groups/cyhist/conversations/messages/1271.

15. Alan Turing, *Programmers' Handbook for Manchester Electronic Computer Mark II*, AlanTuring.org, http://www.alanturing.net/turing_archive/archive/m/m01/M01-030.html.

16. Jack Copeland and Jason Long, "Restoring the First Recording of Computer Music," *Sound and Vision Blog,* 2016, https://blogs.bl.uk/sound-and-vision/2016/09/restoring-the-first-recording-of-computer-music.html.

17. Jack Copeland and Jason Long, "Christmas Carols from Turing's Computer," *Sound and Vision Blog,* 2017, https://blogs.bl.uk/sound-and-vision/2017/12/christmas-carols-from-turings-computer.html.

18. Or, perhaps, these were actually saw-tooth generators. It does seem unlikely due to the nature of binary, digital outputs (see "Fundamentals" section), but I could not find a definitive source documenting the actual synthesis methods used.

lines of melody expressed using square waves.[19] For example, in August 1951, Geoff Hill wrote a 1-bit routine for the CSIR Mk1 that was performed publicly at the Conference of Automatic Computing Machines in Sydney;[20] in 1955, Norman Hardy and Ted Ross rewired a console control lamp to a speaker to perform a rendition of Bach's *Partita No. 3 in E major* BM 701;[21] and, from 1958, service technicians wrote music programs for the German Zuse Z22 computers, one of which was even distributed officially by Zuse.[22]

In the 1960s and 1970s, as technology became both faster and more accessible, programmers began to experiment with alternative approaches. An EP, released in 1970, was created using the DATASAAB D21 and D22.[23] These recordings demonstrate the use of super-fast arpeggios to simulate polyphony (alongside other, perhaps more advanced, audio solutions that were not strictly 1-bit).[24] In 1970, Thomas Van Keuren, a programmer working for the US military in Vietnam, independently employed rapid arpeggiation, programmed on a UNIVAC 1050-II US military computer.[25] When home computers became readily available to hobbyists during the late 1970s, the more complex and widespread routines began to materialize.[26] These routines explored more advanced techniques, such as true 1-bit polyphony (of which more below),[27] and heralded the age of the computer music enthusiast: a prelude to the subsequent chipmusic and demoscene cultures. In order to keep the manufacturing costs low and maintain the affordability of home computers, functional concessions had to be made. Home computers did not have the memory capability to store large amounts of sampled data, thus alternative strategies were devised to include audio in software applications, most frequently video games.[28] Dedicated sound hardware utilized a wide variety of methods such as frequency modulation and wavetable synthesis,[29] however even these could be expensive. As an alternative, PCs were frequently shipped with

19. "Nellie: School Computer," Tomorrow's World, Series 4, BBC Broadcasting Service, February 1969, http://www.bbc.co.uk/programmes/p0154hns.

David Hartley, "EDSAC 1 and After: A Compilation of Personal Reminiscences," *EDSAC 99*, 1999, https://www.cl.cam.ac.uk/events/EDSAC99/reminiscences/.

David Sordillo, "Music Playing on the Pdp-6," Project MAC, 1966, ftp://publications.ai.mit.edu/ai-publications/pdf/AIM-107.pdf.

20. Doornbusch, Paul. "Computer sound synthesis in 1951: The music of CSIRAC." *Computer Music Journal* 28. (2004): 10-25.

21. Norman Hardy, "Music," Stories, 2005, http://www.cap-lore.com/stories/music.html.

22. Zuse, "Programm 47," Zuse Z23 Programmkatalog, 1970, http://www.sol20.org/manuals/music.pdf.

23. Göran Sundqvist, *D21 - In Memoriam - D22* (Vinyl, EP), 1970, Sweden: KDA – RM 5301.

24. Magnus Karlsson, "In memoriam," DATASAAB, 1999, https://blogs.bl.uk/sound-and-vision/2017/12/christmas-carols-from-turings-computer.html.

25. Thomas Van Keuren, "Ebb Tide Played by 1970 Univac Computer (No Sound Card)," January 2015, https://www.youtube.com/watch?v=X6F7qwa5TZg.

26. Software Technology Corporation, "The Music System," *Music System User's Manual*, 1977, http://www.sol20.org/manuals/music.pdf; Sarah Rood, *From Visions to Realities* (Melbourne: Monash University Custom Publishing Services, 2008).

27. Such as The Music System, published by Software Technology Corporation in 1977, which employs true 1-bit polyphony.

28. Melanie Fritsch, "History of Video Game Music," in *Music and Game: Perspectives on a Popular Alliance,* ed. Peter Moormann (Wiesbaden, Germany: Springer, 2012), 11–41.

29. Joey Latimer, "Hit It, Maestro!" *Compute! Magazine*, 1990, https://web.archive.org/web/20140906061745/http://www.joeylatimer.com/pdf/Compute !%20April%201990%20PC%20Sound%20Gets%20Serious%20by%20Joey%20Latimer.pdf.

internal speakers that were attached directly to a processor output pin.[30] Systems such as early models of the ZX Spectrum initially provided the programmer no alternative but to use 1-bit music,[31] and those who did not wish to invest in a sound card for their desktop computer could still experience audio in games and software via the internal speaker.[32] These requirements encouraged games publishers to find the most interesting solutions possible to distinguish their product from the competition and garner more sales.[33] With this attitude of "progression," the industry ultimately discarded 1-bit audio in favor of greater sonic capability and versatility. Development of 1-bit practice was then adopted by hobbyists, those with nostalgia for their 1-bit platform, or those fascinated by the medium. The computers (such as the ZX Spectrum) that booted directly into a programming interface, commonly BASIC,[34] allowed the casual user immediate access to functions that could beep and, with a little clever extrapolation, be coaxed into playing melodies. Those engaging in the emerging demoscene and chipmusic cultures, often having experimented with these simple routines as children or teenagers, pushed what was previously possible. They were aided by Internet collaboration, through competitions, communal sharing of code, and a general enthusiasm for the medium.[35]

FUNDAMENTALS

Instrumental technique in 1-bit music is shaped by its signal, perhaps to a greater extent than other instrumental and musical platforms. To maximize expression and proficiency when using this environment, one must have a basic understanding of the theory and its implementation. The relationship and discrepancies between an ideal, logical pulse wave and its translation to acoustic and analog domains can be exploited to musical effect. As such, practical application and exploitation of these concepts results in unique compositional techniques exclusive to 1-bit music, belying the nature of the environment. Appreciation of how the 1-bit waveform acts conceptually (and psychoacoustically) is necessary for understanding and implementing timbral and sonic interest in 1-bit music.

The sine wave is the most fundamental sonic component of the acoustic domain,[36] but in the digital world the foundational unit is arguably the *pulse wave*. One-bit music is the music of pulse waves: simplistic waveforms with binary amplitudinal resolution. A *waveform*

30. Joakim Ögren, "The Hardware Book," *Compute! Magazine*, 1997, http://www.acc.umu.se/~stric/tmp/hwb13pdf/hwbook.pdf#page=290.

31. Steven Vickers, *ZX Spectrum Basic Programming*, ed. R. Bradbeer (Cambridge: Sinclair Research, 1982).

32. There must have been a significant number of users without sound cards; many games were written to support the PC speaker. To name just a few: *DOOM* (1993), *Prince of Persia* (1989), *SimCity 2000* (1993), and *Total Eclipse* (1988). The companion instruction booklet to the game *Crime Wave* even has instructions on how to connect the PC speaker directly to a stereo system (Access Software Inc., *Crime Wave Instruction Manual*. (Birmingham, UK: US Gold Ltd 1990).

33. Fritsch, "History of Video Game Music."

34. Gary Herman, *Micromusic for the Commodore 64 and BBC Computer* (London: PAPERMAC, 1985); Vickers, *ZX Spectrum Basic Programming*.

35. Fritsch, "History of Video Game Music."

36. Guy Oldham, "Harmonic," in *The Oxford Companion to Music,* ed. Alison Latham (Oxford, UK: Oxford University Press), http://www.oxfordmusiconline.com/subscriber/article/grove/music/50023; Elena Prestini, *The Evolution of Applied Harmonic Analysis: Models of the Real World* (Boston: Birkhäuser, 2004).

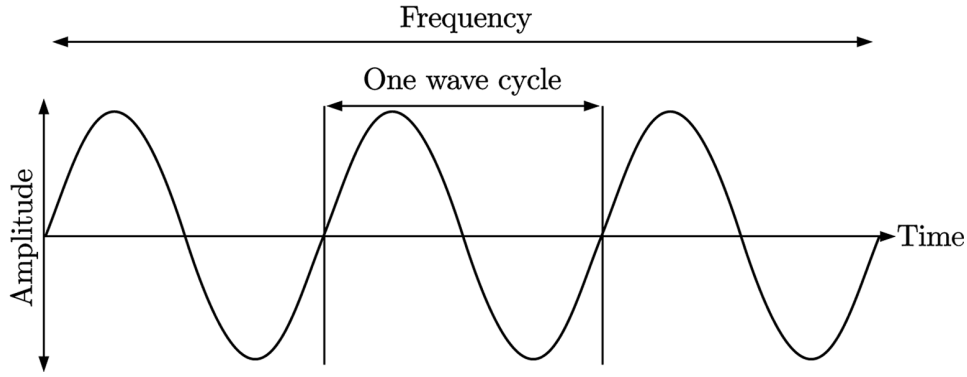

FIGURE 1. Oscilloscope view of a sine wave.

is the shape of an oscillation or vibration, moving through a medium, around a fixed point. A waveform's primary attributes are amplitude and frequency. Plotted on a two-dimensional plane, amplitude is variance on the Y axis and frequency is the addition of time on the X axis (Figure 1).[37] Whilst a sine wave's amplitude can be traced continuously across the Y axis, in a 1-bit environment, for any given point in time, each step of the waveform can be only one of two states: logical high or low, on or off.

The pulse, or rectangle wave (described as such due to the geometry of the waveform's graphed appearance; see Figure 2), is a non-sinusodial periodic waveform defined by series of instantaneous switches between two distinct logic levels.[38] Often, these quanta are electronic oscillations between two voltages (usually positive and ground); however, pulse waves can exist as a longitudinal pressure wave or as an abstract mathematical function. The period between two amplitudinal events is considered a *pulse*. Regular periodicity of pulses (a *pulse train*) determines its *frequency*: the rate of repetition in any periodic quantity.[39] Discernible frequency is required for texture and pitch coherency, whereas random distributions of pulses result in unpitched audio noise, approximating white noise.[40] This definition is important as it suggests the first instrumental capability of the 1-bit pulse: the ability to produce both percussive (unpitched) and melodic (pitched) sonorities. The 1-bit musician must simply change the order of pulses from regular to unordered to generate two vastly different textures. The duration of a pulse event's *mark* (high) time is referred to as its *pulse width* (indicated by W_1 in Figure 2, while the *space* (low) time is indicated by W_2).[41] The relationship between the pulse width and the total cycle duration can be expressed as either a ratio or a percentage, known as the waveform's *duty cycle*. A duty

37. Howard E. Haber, "How to Add Sine Functions of Different Amplitude and Phase," 2009, http://scipp.ucsc.edu/~haber/ph5B/addsine.pdf.

38. Herman, *Micromusic for the Commodore 64 and BBC Computer*, 28–29.

39. John Borwick, "Frequency," in *The Oxford Companion to Music*, ed. Alison Latham. (Oxford, UK: Oxford University Press), http://www.oxfordmusiconline.com/subscriber/article/opr/t114/e2693.

40. Alison Latham, ed., "White Noise," in *The Oxford Companion to Music* (Oxford, UK: Oxford University Press), http://www.oxfordmusiconline.com/subscriber/article/opr/t114/e8240.

41. Herman, *Micromusic for the Commodore 64 and BBC Computer*, 22–23.

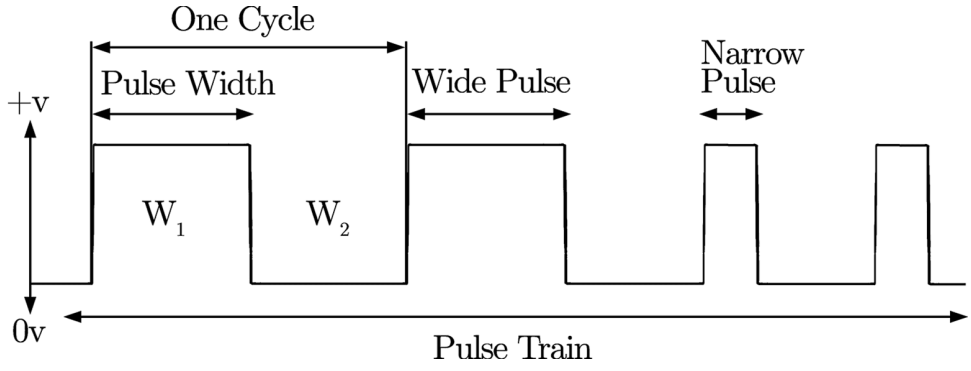

FIGURE 2. The topography of the pulse wave.

cycle of 50% (a ratio of 1:1) would indicate a pulse width with equal mark and space time. Though there *is* technical differentiation between the definitions of pulse width and duty cycle, both terms are often used synonymously in the chipmusic community, referring to the ratio between the waveform mark and space time.[42] Whilst theoretically infinite in variation, the maximum number of unique duty cycles is limited by both hardware fidelity and human inability to discern a waveform's *phase*.[43] A duty cycle of 75% (3:1) cannot be aurally distinguished from a duty cycle of 25% (1:3), as these are considered *phase inversions* of each other.[44] The waveforms are offset by 180 degrees; the mark time of the 75% waveform corresponds to the space of the 25% waveform, and these are perceptually identical when inverted (Figure 3). Due to this effect, all similar widths above and below 50% are timbrally identical to the human ear (by means of: $50\% \pm n, n < 50\%$).

As a practical example, Listing 1 (written in C pseudo-code) generates a single square wave, demonstrating perhaps the most elementary 1-bit tone generator and its composition. Here we can see a scripted version of the aforementioned list of basic operations required for tone generation. The desired frequency can be set with the `frequency` variable. The timer `pitch_counter` counts down from the set `frequency` value to zero. The main loop of the program checks to see if the counter has reached zero. When that happens and `pitch_counter = 0`, the output is alternated (from on to off, or vice versa) and the timer is reset. The `pitch_counter` begins counting down again, ready for the output to be switched again. As long as `frequency` remains constant, the intervals between alternating on and off are equal, and so the routine produces a pulse wave of equal high and low durations: a square wave.

42. Herbert Weixelbaum, "Game Boy Sound Comparison," Game Boy Music, June 2019, http://www.herbertweixelbaum.com/comparison.htm.

43. Phase is the position of a point of time on a waveform cycle, subdivided into 360 degrees of possible offset from the origin ("Phase," National Institute of Standards and Technology, September 2016, https://www.nist.gov/time-and-frequency-services/p).

44. Steve Lakawicz, "The Difference Between Pulse Waves and Square Waves," Research in Game Music, Classical Gaming, May 2012, https://classicalgaming.wordpress.com/2012/05/15/research-in-game-music-the-difference-between-pulse-waves-and-square-waves/.

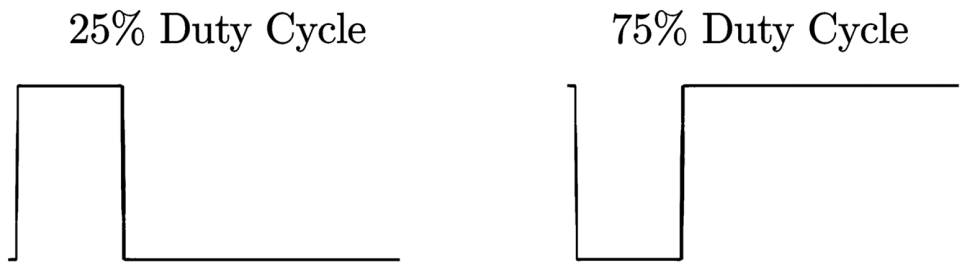

FIGURE 3. Duty cycles of ratios 1:3 and 3:1. Notice the inverse polarity of the mark and space time.

```
if(--pitch_counter == 0)
    pitch_counter = frequency;
    output ^= 1;
}
```

LISTING 1. A square wave generator written in C pseudo-code. Note that the `pitch_counter` and `frequency` variables actually represent half the frequency; as decrementing to zero changes the amplitude, two complete cycles are required to generate a complete waveform.

TIMBRE

Discussion and analysis of aural techniques in reference to 1-bit theory *alone* exclude the dependencies between the theoretical and actual waveform; the 1-bit paradigm does not perfectly translate from the conceptual to the acoustic domain. The digital-to-analog converter (DAC) and electronic circuitry can misrepresent signals, subjecting pulse waves to deformations such as *ringing*, *rounded leading edges*, and *overshoot*,[45] demonstrated in Figure 4.[46] Even internally, a microcontroller may not update all bits in a register at the same time, causing erroneous values.[47] Alternatively, deficiencies in the frequency spectrum of the audio technology may result in an alteration of the waveform. All of these distortions alter the intensity and weighting of the signal's harmonics: component sinusoidal tones arranged in successive integer multiples of the first harmonic (known as the *fundamental*).[48] This arrangement is known as the *harmonic series* and is responsible for the perception of timbre: the identifying characteristics of a sound.

Figure 5 illustrates the relationship between timbre and waveform. The first waveform demonstrates the "ideal" square wave (a pulse wave with a duty cycle of 50%), with a (comparatively) excellent frequency response, as indicated by the intensity of the dark bands in the spectrogram view.[49] A *spectrogram* is a visual representation of the harmonic components of a sound. The pictured bands represent individual harmonics, where the lowest

45. R.G. Middleton, *Know Your Square Wave and Pulse Generators* (Carmel, IN: H. W. Sams, 1965).

46. Butler, "Waveforms Using the Cathode Ray Oscilloscope"; E. G. Louis, "Practical Techniques of Square-Wave Testing," Radio & TV News, RF Cafe, July 1957, http://www.rfcafe.com/references/radio-news/practical-techniques-square-wave-testing-july-1957-radio-tv-news.htm.

47. Ken C. Pohlmann, *Principles of Digital Audio*, 6th ed. (New York: McGraw-Hill, 2011), 86.

48. Lloyd Butler, "Waveforms Using the Cathode Ray Oscilloscope," Waveform and Spectrum Analysis, June 1989, http://users.tpg.com.au/users/ldbutler/Waveforms.htm; Oldham, "Harmonics."

49. Louis, "Practical Techniques of Square-Wave Testing."

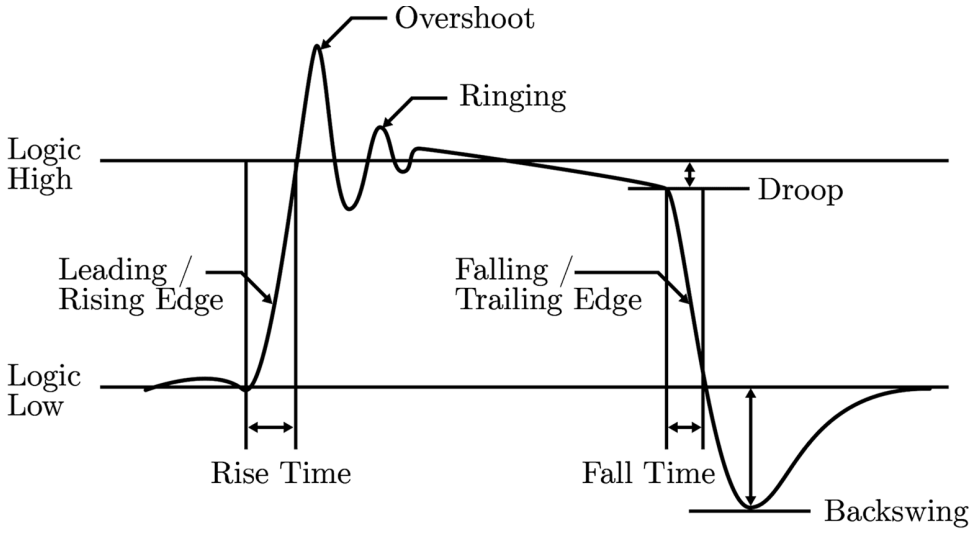

FIGURE 4. A diagram of possible distortions and deformations to the pulse component of a rectangle wave. Ringing is oscillation after the leading edge, often proceeded by overshoot, where the signal's amplitude increases beyond the logical high level. These distortions will indicate that the signal's harmonic image deviates from that of an ideal pulse wave.

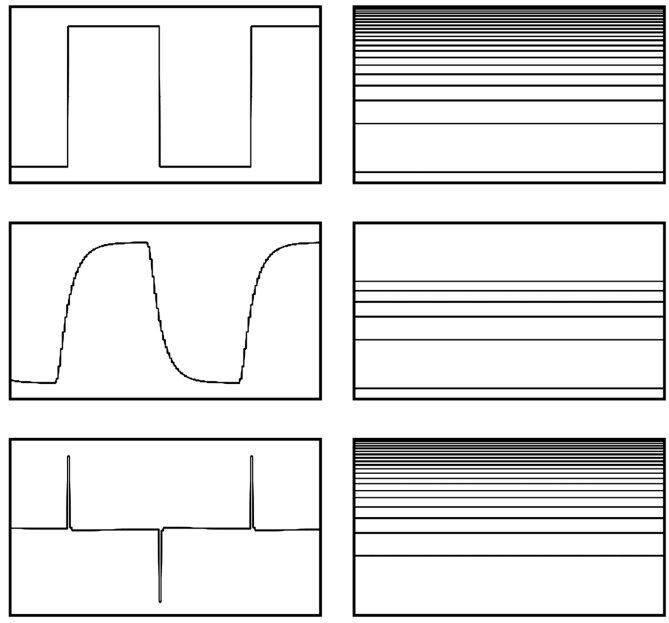

FIGURE 5. The waveform (left) and frequency spectrum (right) view of three permutations of square wave over time. From top to bottom: excellent response, poor low-frequency response, poor high-frequency response. The scaling of the spectrogram is logarithmic—skewed to align closer with human perception of pitch and severely thresholded to the loudest harmonics to clearly demonstrate the relationship between waveform and timbre. Generated at a sampling rate of 44100 Hz with Image Line's 3x Osc.

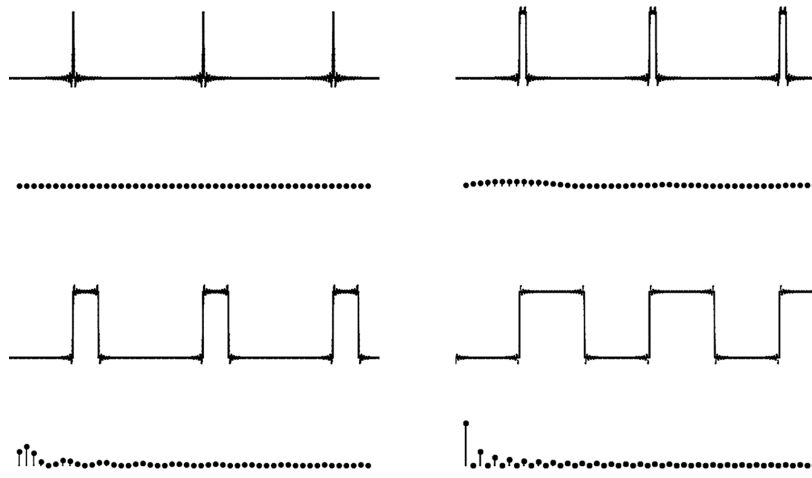

FIGURE 6. A waveform (top) and harmonic (bottom) view of four duty cycles, getting progressively wider with each image (left to right). Produced with Paul Falstad's Fourier Series Applet.

band is the fundamental. The effect of rounded leading edges to the second waveform has a very obvious timbral effect when compared to the ideal waveform's spectrogram image. Aurally, this will "dull" or "muffle" the sound, as if heard through fabric or a wall. The third shows a deficiency of the lower frequencies, which will have "brighter" yet "thinner" quality due to a weaker bass response than the other two examples. Therefore, deformations of the waveform in practice, such as those in Figure 4, will alter the timbral output. Thus, when approaching practical 1-bit composition, one must consider the method of physical implementation by which the listener will experience the composition. Ultimately, any software generation routine will be subject to timbral alteration by the physical method of sonic propagation, including the enclosures housing the electronics. As a practical example, Tim Follin's work for the ZX Spectrum game *Chronos* does not account for this, sounding coherent in emulation but unintelligible on some systems.[50]

In addition to the timbral artifacts of external circuitry, timbre can be more noticeably and predictably altered in software by adjusting a pulse wave's duty cycle. As the duty cycle decreases, narrower pulse widths are progressively quieter than wider widths in the lower harmonics, with 50% being the *perceptually* loudest width possible. Figure 6 depicts four duty cycles, increasing in pulse width from left to right, top to bottom. Shown below the waveform is a representation of the timbre—the integral harmonic components and their respective amplitudinal intensities via Fourier transform.[51] Starting with the thinnest, top-left pulse wave, the height (representing amplitude) of the fundamental is greater in intensity as the sequence of images increases, demonstrating the reduction of the apparent

50. This is humorously observed in a 1987 review in *Crash* magazine, describing the music as "a strange bit of title sound (rather than music)" (Paul, "Chronos," *Crash* 41, 1987, https://archive.org/details/crash-magazine-41).

51. The Fourier transform is a series of operations that dismantle a continuous function of time into its harmonic components. Therefore, any periodic signal can be reconstructed from sine waves with frequencies that are multiples of the fundamental.

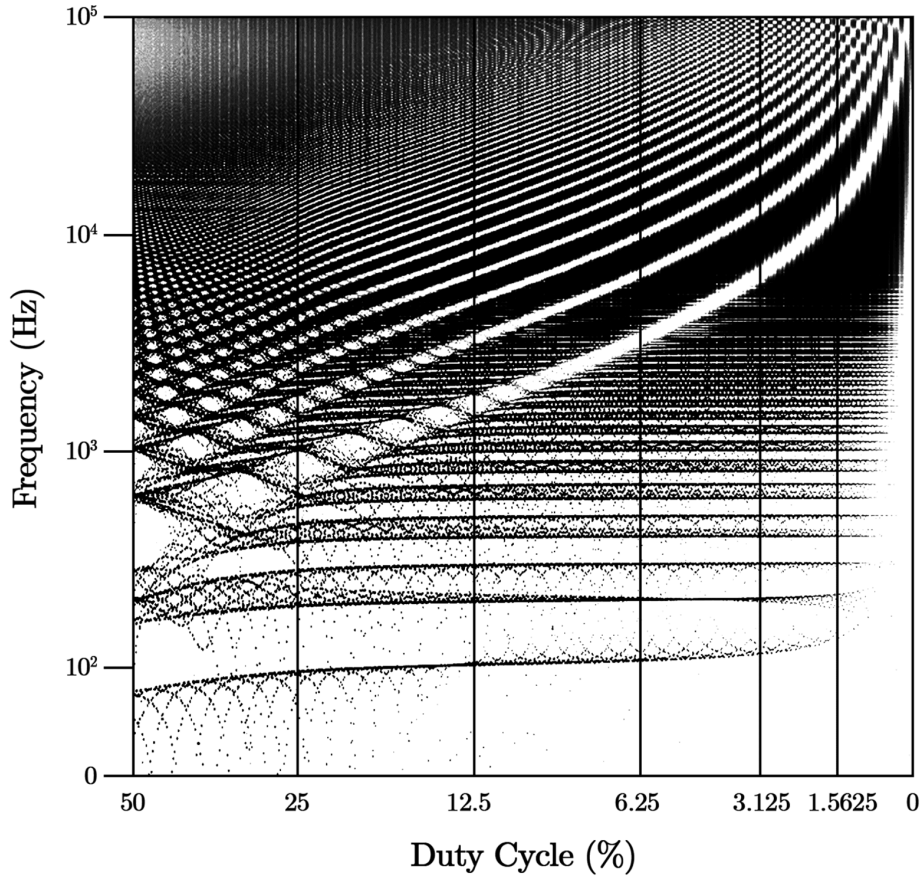

FIGURE 7. A spectrogram view of the pulse width example, where a square wave is progressively narrowed in pulse width over time. The duty cycle is decremented (nonlinearly) from 50% through to 0%. Significant widths are identified by the vertical markers. The illustration here demonstrates the changes to the frequency components at duty cycles between 50% and 0% where brightness (or lack thereof) indicates the frequency's power; stronger frequencies are represented by darker lines. The spectrogram was generated using Image-Line's Edison software. The generated sample rate was 214000 Hz. See `pulse-width-sweep.wav` for an audio example.

"high-pass" effect and increased emphasis of the signal's "bass" range. This change is not discrete as the images might imply, but continuous—as illustrated in Figure 7—and is a product of the harmonic nature of the signal.[52]

Even though the amplitude is a constant 1-bit waveform, the narrower pulses provide a way of varying volume. These narrower pulses have incrementally less power *overall* to the listener; as the duty cycle approaches 0% (or, by inversion, 100%) the perceptual volume decreases with it, even though the amplitude remains the same. This effect is not a consequence of the reduction of the pulsing signal's actual, electronic, or kinetic power. Instead, the reduction in volume is a product of *bandlimiting*—the effect whereby frequencies beyond

52. Steven Smith, *The Scientist and Engineer's Guide to Digital Signal Processing*, San Diego: California Technical Publishing, 1997, 243–260.

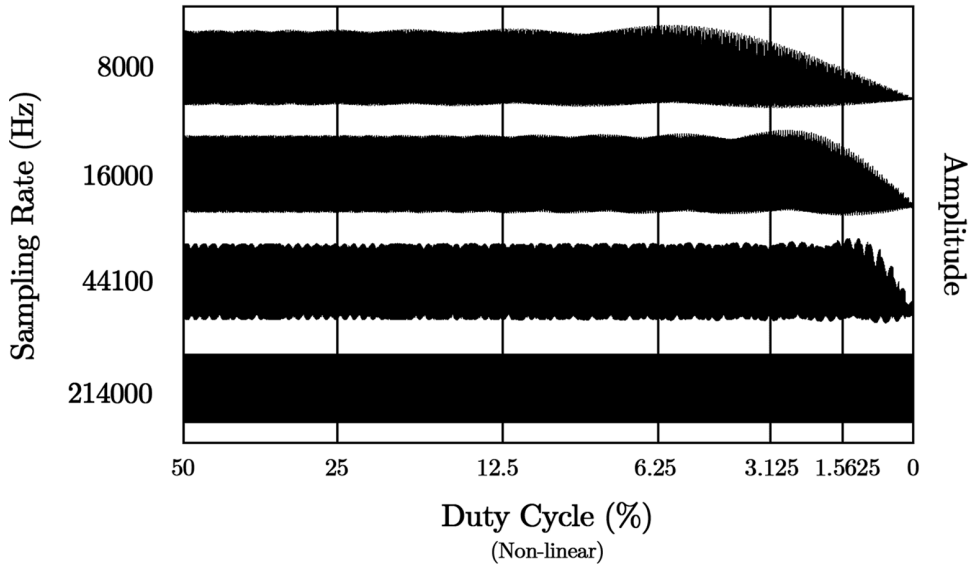

FIGURE 8. Four waveforms of the pulse width example (`pulse-width-sweep.wav`) at different sampling rates. Each has been "downsampled" (using Image-Line's Edison audio editor) from the raw file. Generated at 214000 Hz. Downsampling cuts those harmonics faster than half the sampling rate, thus reducing the power of waveforms with stronger high-frequency harmonics.

a particular value are not heard. The situation is akin to the digital sampling of analog sound: when the resolution of the sampling is too low to capture high-frequency components of the sound, they are not registered. One-bit music is *acousmatic*: it is presented exclusively through speakers; it cannot be generated naturally. As such, because the frequency response of a speaker is limited by how fast the cone can physically move, higher frequencies will not be replicated by the diaphragm.

As we saw in Figure 5, thinner pulses are constructed from more powerful high-frequency harmonics than lower ones. Accordingly, as the pulses get thinner, with extremely small or extremely large duty cycles, these higher frequencies increasingly fall outside the limits of what can be replicated by the speaker. Since these elements are not present, the result is a reduction of the waveform's overall power.

This volume-altering effect is illustrated in Figure 8, which uses slower sampling speeds to illustrate the restriction on the waveform, leading to a loss of amplitude. At a fast sampling speed, amplitude does not decrease when the pulse width is narrow, but when the sampling rate is decreased, however, this same waveform appears to, sympathetically, decrease in amplitude.

Figure 8 has been downsampled in software, meaning it is a conceptually "perfect" bandpass,[53] which does not translate to the chaotic, noisy, and unpredictable natural world; so how does the reduction in amplitude relate to perception? Filtering (in this

53. I have read, anecdotally, that software low-pass filters are often implemented imperfectly and can actually boost around the cut-off frequency ("Why Not Always Cut the 20–30 Hz Range?" KVR Audio, https://www.kvraudio.com/forum/viewtopic.php?f=62&t=313807&sid=7da7241355268614d8690cad3702890b). I have found this to be true with

case, *lowpass filtering*[54]) is occurring as the waveform is both propagated and sensed: bandlimited by real, physical media at every stage of transmission. Additionally, even if the replication of higher frequencies were perfect and transmitted through an ideal, theoretical medium of infinite bandwidth, the upper limits of human perception is, unavoidably, around 20 kHz.[55] Thus the appearance of amplitudinal change is caused by a conceptually perfect model translated through an imperfect environment and human response.

This phenomenon has multiple musical implications. First, due to the distributions and intensities of harmonics present in different pulse widths, some widths are more suitable in particular traditional instrumental roles than others. For example, as human hearing is less sensitive to frequencies below approximately 1 kHz[56] (and increasingly so as the frequency decreases), those pulse widths with stronger low-frequency partials are better suited to material that would conventionally employ bass instruments. Furthermore, as it is possible to change the perceptual loudness of a voice (even though only two amplitudinal states are conceptually and *abstractly* possible),[57] the composer has the surprising ability to introduce dynamic variation into their composition. Of course, this comes with the sacrifice that timbre and volume are concomitant—in that, if one wishes to alter dynamics, one must also inextricably alter the timbre. This is less noticeable at larger widths, but it becomes discernible at widths of 5% or lower, visually demonstrated in Figure 7. If one looks at the center of the spectrogram, the tapering of intensity of the lowest harmonics should be evident as the width approaches 0%. It is at these extremes that a reduction in volume is perceptible. As the volume decreases, the bass harmonics perceptually recede first, sounding ever more "reedy" or "nasal" in texture. In contrast, those duty cycles approaching 50% sound progressively more "rounded" and "clarinet-like"[58] (for lack of more specific terms). How the synthesis is *compositionally* employed plays a significant role in how apparent (and disparate) each voice's timbre-volume pairing is. Those 1-bit compositions in which thin pulse widths are exclusively utilized sound texturally homogeneous; the apparent changes in amplitude are more convincing as genuine changes in volume. Figure 9 aurally demonstrates this effect. The example is split into two arpeggios, equal in both frequency and time. Both arpeggios are composed of four dotted eighth notes, each of these subdivided into sixteenth notes.

FL Studio's Fruity EQ 2, so I cannot guarantee that, when downsampling in software, I am not inadvertently introducing errors.

54. Smith, *The Scientist and Engineer's Guide to Digital Signal Processing*, 261–276.

55. Smith, *The Scientist and Engineer's Guide to Digital Signal Processing*, 351–353.

56. Erich Carr Everbach, "Noise Quantification and Monitoring: An Overview," The Science Building Project, 2000, http://www.swarthmore.edu/NatSci/sciproject/noise/noisequant.html.

57. Here the use of the word *voice* refers to a distinct instrumental line with its own sonic identity. As virtually all applications of 1-bit music are generated in software, there are no fixed "channels." We can consider these voices as particular entities that are given individuality by their musical application. Although no predefined channels exist, channels are often implemented in software. This approach has been taken in the various code examples used throughout this article.

58. The perceived similarity between a 50% pulse wave and a clarinet may be due to the similar spectral images; both clarinets and square waves produce, overwhelmingly, odd harmonics, as opposed to the comparatively quiet even partials (Carl Rod Nave, "Clarinet Waveform," 1998, http://hyperphysics.phy-astr.gsu.edu/hbase/Music/clarw.html).

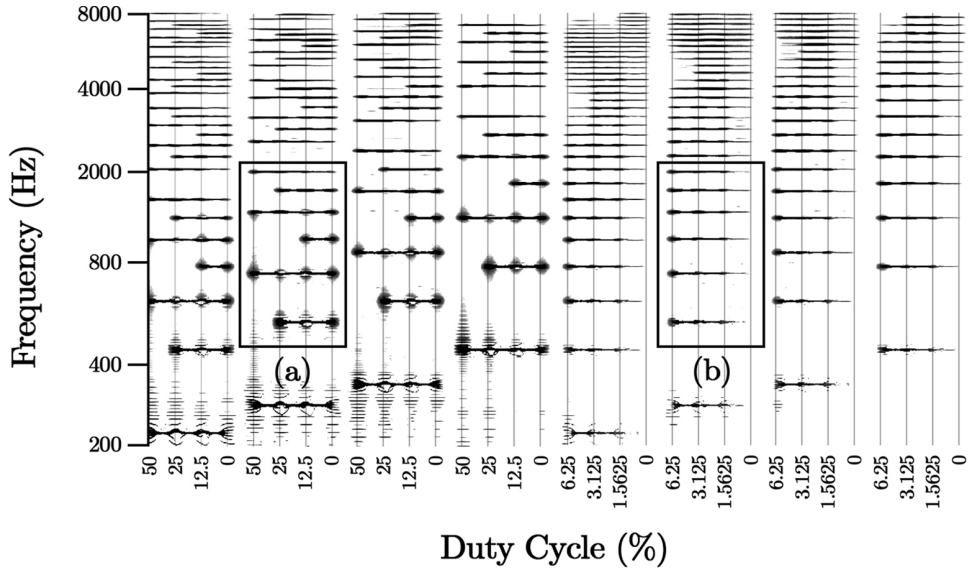

FIGURE 9. Spectrogram view of two identical ascending sequences demonstrating how the timbral character of a pulse wave, in relation to the duty cycle half its width, changes as the starting pulse width is reduced. Each arpeggio consists of dotted eighth notes subdivided into four sixteenth notes of fixed frequency (one of silence), differing only in duty cycle. Each sixteenth note is half the duty cycle of the one before it. The first arpeggio starts each note at 50%, the second at 6.25%. The visualization was created using Image-Line's Edison audio editor, and the example was generated at 60000 Hz. See `timbre-vs-volume.wav` to listen to the original audio.

Every sixteenth note is half the duty cycle of that before it. The first sixteenth note starts at 50% (then 25% and 12.5%), the second at 6.25% (then 3.125% and 1.5625%). Despite both sets of duty cycles having identical ratios between elements (being ratios of one another to the power of two) it is only in the second set, at narrower widths, that the timbre no longer appears to transform over the duration of the note; instead the apparent volume decreases. Both annotations *(a)* and *(b)* in Figure 9 show the first six partials above the fundamental for duty cycles 50%, 25%, and 12.5%, and 6.25%, 3.125%, and 1.5625%. The harmonics highlighted in *(a)* (those that have the greatest effect on the human perception of timbre) alter dramatically between the duty cycles; thus, over the duration of the note, these changes are heard as timbral—more a series of instrumental transformations. Comparatively, those partials presented in *(b)* remain consistent between the pulse width changes, but recede in intensity (at least in the lower harmonics), perceptually varying less in timbre but more in volume. This behavior is a product of the distribution of harmonics in any given pulse wave; as we can see in Figure 7, the spectral image can be described by $m = \frac{1}{pw}$, where pw is the given duty cycle and every mnth harmonic ($n = 1 \ldots k$) will be missed.

Therefore, as the duty cycle is halved, so too is the total number of missing harmonics; the timbre becomes ever more timbrally congruous and sonically cohesive to the human listener. A thin pulse is, practically, a high-passed saw wave: to the human observer, nearly all integer harmonics are present, but in the case of the pulse, with a progressively weaker

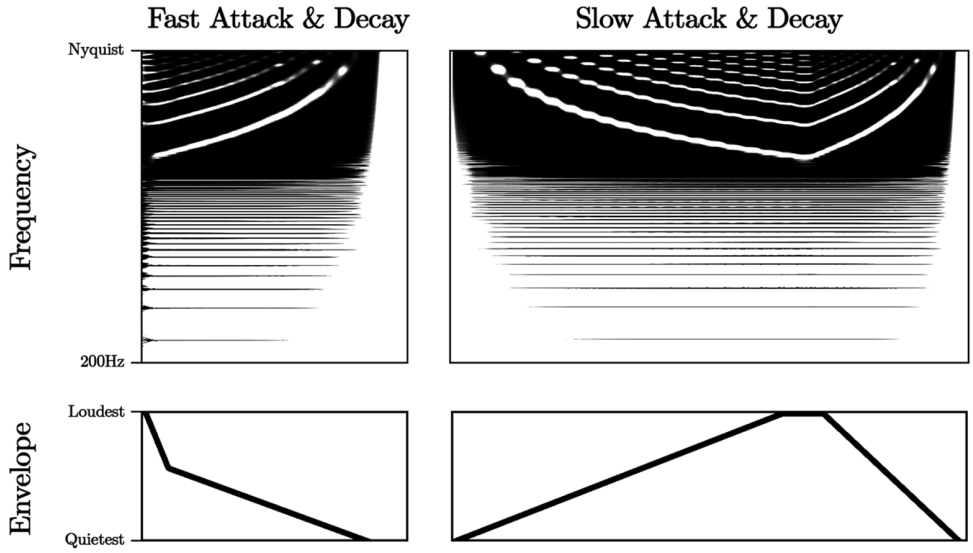

FIGURE 10. Two spectrograms visually demonstrating 1-bit volume enveloping. The left example depicts a quick attack, starting at the maximum value, decaying to a medium, brief sustain level, and then moving quickly to silence. The right shows a ramp from zero to sustain, then again a decay to silence. These do not show true changes in amplitude but, instead, in pulse width. The example was generated at 214 kHz. See `volume-enveloping.wav` to listen to the original audio.

bass response.[59] Consequently, the 1-bit composer cannot employ a quiet voice with a strong bass component, as a reduction in volume will result in a considerable alteration of timbre and a perceptual separation of instrumentation.

The phenomenon in Figure 7 allows the 1-bit composer access to volume *enveloping*, or transforming a sound's volume over its duration,[60] while maintaining pitch, which increases the pulse wave's repertoire of instrumental expression. Conventional ADSR[61] enveloping (the modulation of a sound's volume over four periods: attack, decay, sustain, and release) is not only theoretically possible but implemented in early 1-bit soundtracks such as *The Sentinel*[62] and *Chronos*[63] on the ZX Spectrum—as well as many 1-bit routines.[64] Figure 10 depicts a simple implementation of ADSR enveloping on a 1-bit pulse wave, demonstrating a fast attack and decay, alongside a contrasting slower attack and decay. The "trick" is to keep the maximum fade width to approximately 6.25% and where the initial missing harmonic is beyond the first ten or so in the harmonic series (where variations are most noticeable) so that the fade is aurally congruous and mutates

59. Barb Bland, "Making Complex Waves," 1999, http://hep.physics.indiana.edu/~rickv/Making_complex_waves.html; Smith, *The Scientist and Engineer's Guide to Digital Signal Processing*, 261–276.

60. Tim Rutherford-Johnson, ed., "Envelope," *Oxford Dictionary of Music,* 6th ed (Oxford: Oxford University Press, 2013).

61. David Strange, "ADSR Envelope Generator," *Electronic Soundmaker*, no. 2 (1984).

62. *The Sentinel*, 1987, Software Creations/Firebird Software Ltd., ZX Spectrum.

63. *Chronos*. 1987. Mastertronic Ltd., ZX Spectrum.

64. *Orfeus Music Assembler*, 1990, Proxima Software, ZX Spectrum; ZX-3, 1991; Ján Deák/Elektronika, ZX Spectrum.

more in perceptual power than timbre. Additionally, as halving the duty cycle concomitantly doubles the change in perceptual volume, one may wish to implement a compensatory nonlinear fade so that the apparent volume change is *perceptually* linear.[65]

The emphasis placed on narrow duty cycles does not imply that movement between wider duty cycles is unpalatable. Envelopes that traverse wider widths modulate timbre instead, as heard towards the beginning of the example in Figure 7, between 50% and approximately 10%.[66] This effect can be employed to add further expression and individualism to discrete voices, approximating how acoustic instruments may change their spectral image over time.[67] These changing harmonics are considered *transients* and the entire envelope is recognized as an instrumental gestalt by the human listener. The `pwm-enveloping.wav` and `pwm-enveloping-scale.wav` audio examples are a modification of the routine used in Figure 10. This implementation ignores the previously imposed maximum duty cycle, allowing the envelope access to larger pulse widths.

The spectrum of pulse widths encompasses a great variety of timbres, allowing the creation of individual characteristic voices. Even though all these waves are generated via identical 1-bit parameters, the movement of pulse width over time forms a voice's instrumental identity. Musical examples of *pulse width modulation* (PWM)—the alteration of duty cycle over time for timbral effect—are numerous and the approach is certainly not unique to 1-bit practice.[68] One-bit is, however, distinct in its peculiar use of pulse width *extremity*—the utilization of very thin widths during a PWM sweep, producing a continuous blend of timbre to volume changes.

As pulse width does not affect pitch, any operation to alter duty cycle must occur within the duration of a wave cycle. Using Listing 1 as a starting point, two conditions can be added to toggle the output depending on the value of the `pitch_counter` variable. In Listing 2, this value is represented by the `waveform` variable. The `waveform` value can be set based on a division of the frequency. Dividing `pitch_counter` by two will result in a pulse wave of 50% duty cycle, dividing by four a duty cycle of 25%, dividing by eight yields 12.5%, and so on.

```
waveform = frequency / 4;

if(--pitch_counter == 0)
    pitch_counter = frequency;

else if(pitch_counter <= waveform)
    output = 1;
```

65. Daniel J. Levitin, *This Is Your Brain on Music: The Science of a Human Obsession*. (New York: Dutton, 2006), 69–70.

66. Of course, it should be noted that, as duty cycles beyond 50% are phase inversions of those below, sweeping to widths larger than 50% will aurally reverse the direction of modulation, appearing to "bounce" back when reaching 50%.

67. Levitin, *This Is Your Brain on Music*, 53–54.

68. For example, the Commodore 64's SID audio chip allowed composers access to continuous changes in pulse width, demonstrated in pieces such as Peter Clarke's famous *Ocean Loader 3* music (Peter Clarke/Ocean Software, 1987, Commodore 64 Loading Software). Some 1-bit examples include Brian Marshall's soundtrack to *Last Ninja 2* (System 3 Software Ltd., 1988, ZX Spectrum) and Mister BEEP's "Chromospheric Flares" (ZX Art, 2014, https://zxart.ee/eng/authors/m/mister-beep/chromospheric-flares/).

```
else if(pitch_counter >= waveform)
    output = 0;
```

LISTING 2. An example of a variable pulse width generator in C pseudo-code. This waveform variable could be simply modified by changing the divisor, but, as it currently exists, the example will produce a fixed duty cycle of 25%. The thinnest width possible is dependent on the size of the `pitch_counter` and `frequency` variables, with higher values yielding thinner available widths. The thinnest width will always be a `waveform` value of 1, but the proportion of 1 to the `frequency` value changes depending on the size of `frequency`. For example, 1 is 10% of 10, but 0.5% of 200.

POLYPHONY

On first inspection, it would seem impossible for simultaneous voices to be expressed by a 1-bit waveform. For a signal to carry two frequencies, concurrently traveling through the same medium, one would expect a superposition of one waveform on the other: a summation resulting in phase addition (or subtraction) where the interacting waveforms constructively or destructively interfere.[69] This cannot happen in a 1-bit environment; because the waveform may exist in either of two quantized states, additional signals will fail to be represented. Figure 11 illustrates this effect with the addition of two 1-bit waveforms and the subsequent conversion back to a 1-bit composite. The product of this combination will be noisy, incoherent, and not clearly resemble either of the original frequencies. One can consider this merger equivalent to extreme distortion, or "clipping"[70] (where 0 dBFS is one bit above $-\infty$ dBFS, or DC zero).

The solution to packing multiple frequencies into a binary DC offset waveform is to reduce the rate of peak interactions and minimize distortion. This can be achieved by reducing the pulse width to avoid overlap (Figure 12). Very thin pulses ($\approx < 10\%$) will allow the majority of waveform interactions to be trough to trough, or peak to trough, which does not affect signal quiddity. When two peaks *do* eventually overlap, there will still be unavoidable distortion, but the regularly of these interactions will be so infrequent as to retain the identity of the union. Therefore, we can imagine successfully merged 1-bit signals as the application of the logical OR operation on highly narrow pulse widths.[71] The OR operator returns true if either or both operands are true; if not, it returns false. If a signal high event is present in any individual voice attempting to be mixed, it will be represented in the union, and if no voices have high events, the resultant signal will be low. This solution is known as the *pin pulse method* (PPM) or pin pulse technique.[72]

69. Daniel A. Russell, "Acoustics and Vibration Animations," The Pennsylvania State University, 2014, https://www.acs.psu.edu/drussell/demos/superposition/superposition.html.

70. J. Corey, *Audio Production and Critical Listening: Technical Ear Training* (Abingdon, UK: Taylor & Francis, 2016).

71. Often routines employing this type of mixing will use the "exclusive or" (XOR) operation. Although the truth tables for both operations slightly differ, the resultant sound is very similar. I have generally found OR mixing to be more coherent (and more aurally palatable) when duty cycles are increased.

72. utz, "Tutorial: How to Write a 1-Bit Music Routine," 1-Bit Forum, July 2015, http://randomflux.info/1bit/viewtopic.php?id=21.

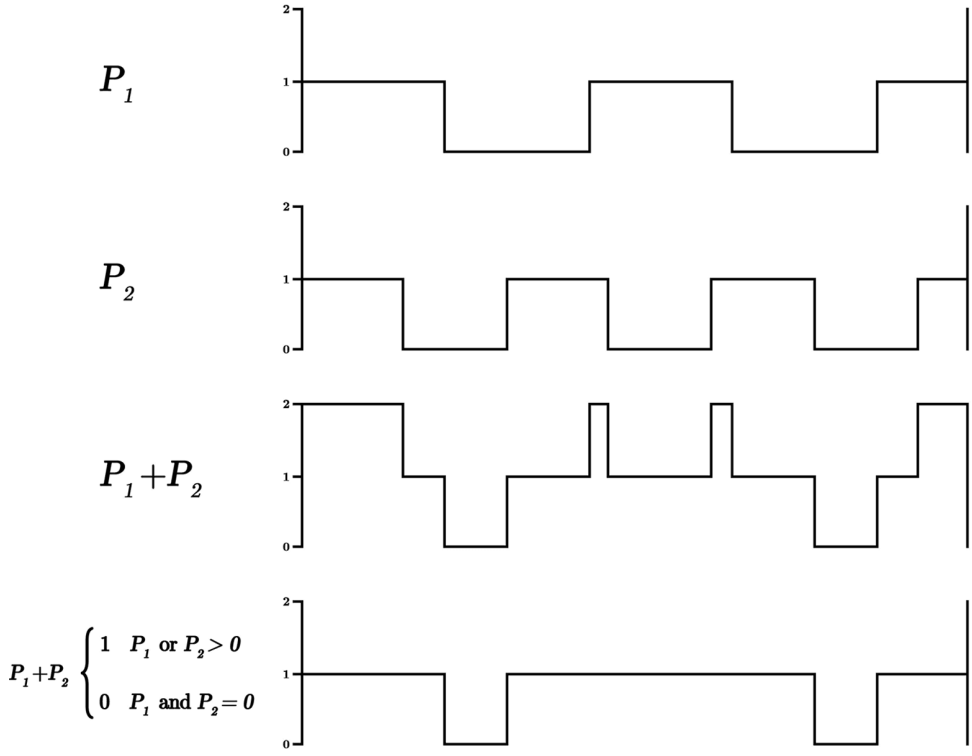

FIGURE 11. The summation of 1-bit signals is not the same as that of sinusoids. As 1-bit waveforms are either of amplitude zero or one, they can be considered to be DC offset, where the amplitude is displaced such that the trough is always DC zero. Resultantly, there is never any wave subtraction, only addition. The reader may realize that this behavior is equivalent to a logical OR operation—this observation will be useful later.

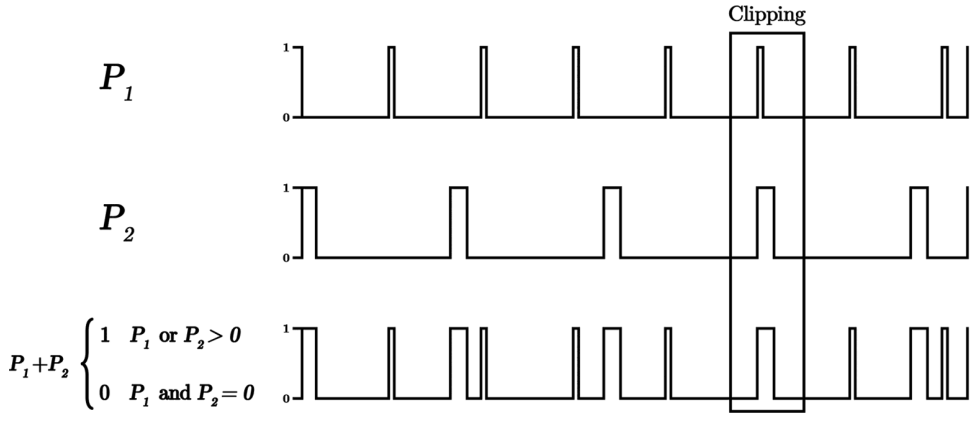

FIGURE 12. The summation of 1-bit signals via the pin pulse method. The identity of both P_1 and P_2 is clearly retained in the aggregate. The highlighted area indicates where clipping has occurred (the peaks have collided and exceeded the amplitudinal resolution); whilst this is unavoidable, it happens so rarely it does not affect the synthesis.

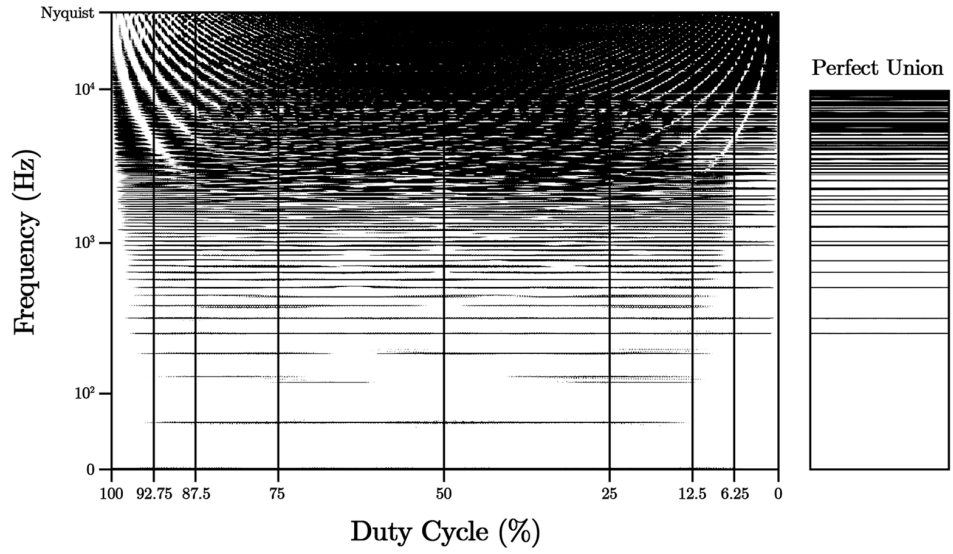

FIGURE 13. A spectrogram demonstration of two waveforms combined at an interval of a major third. In the example both voices are modulated from 100% through to 0% duty cycle. A perfect union (of saw waves—a useful reference as they contain all the integer harmonics) is shown for reference; note how the final combination of pulses at (≈6.25%) is almost identical to a true summation. The visualization was created using Image-Line's Edison audio editor and the example was generated at 64100Hz. See `pin-pulse-method.wav` for an audio example.

Figure 13 is a visual representation of two 1-bit waveforms summed in this way, at an interval of a major third. Toward the end of the modulation (between 12.5% and 6.25%), notice how the harmonics become less "noisy," more stable and accordant with the expected spectral image (the image that would be seen if the combination were at a higher amplitudinal resolution, shown to the right for reference). Surprisingly, the final combination of harmonics is preserved across the entire image (see the two bottommost lines at duty cycle 6.25%), meaning that the combination is largely present across all pulse widths, just with varying levels of additional noise. This "gritty" texture, whilst not a perfect representation of the ideal signal, can be employed to musical effect and, if the composer does not find a certain level of distortion unpalatable, can still communicate coherent polyphony.[73] This noise can be used to "thicken" a 1-bit piece, busying the soundscape to apply texture. Figure 14 utilizes this gradual distortion to pleasing effect; sweeping an add9 chord from duty cycles 0% to 100% to produce a sound not dissimilar to incrementally increasing the input gain on an overdrive guitar pedal.[74] Figure 14 highlights another benefit of this

73. It should be noted that if the waveform was inverted so that 0 was 1 and 1 was 0, Figure 13 would reverse so that it became progressively *more* distorted. I assume in this article that 0 is off and 1 is on, following general convention. There is no "chirality" so to speak; the speaker may oscillate between *any* two states and the theory would still hold. In the event of reversing the signal logic level, Figure 13 would sound most consonant toward 100% and distorted approaching 0%.

74. G. Davis, G. D. Davis, R. Jones, and Y.I. Corporation, *The Sound Reinforcement Handbook* (Milwaukee, WI: Hal Leonard, 1989).

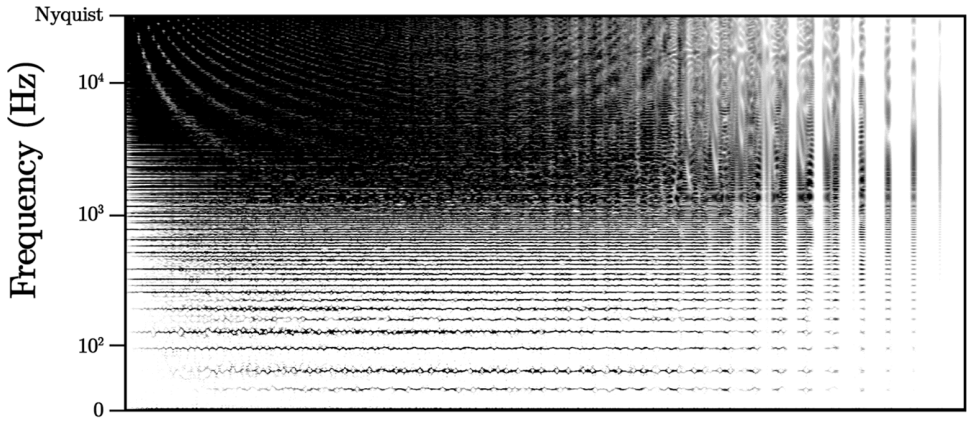

FIGURE 14. A spectrogram view of five 1-bit waveforms combined using PPM and widened from 0% to 100% duty cycle. The chord gradually becomes increasingly distorted as it decays. Interestingly, the example has some noticeable periodic interference, known as beating, intermittently boosting the root of the chord. This beating is more severe toward the end of the spectrogram, where there is oscillation between "silence" (the voices at this point are at 100% duty cycle) and sound. The visualization was created using Image-Line's Edison audio editor, and the example was generated at 64100 Hz. See `pin-pulse-chord.wav` for an audio example.

technique: the pin pulse method is so effective that numerous frequencies can squeezed into the same signal; simply apply the logical OR operation to any new waveform onto the existing composite. As a general rule, the more pulse "channels," the thinner the pulse widths must be before distortion; with every additional signal, the chance of a peak–to–peak interaction is ever more likely.

To implement PPM in software, for each sample one must execute a bitwise OR operation (the '|' character in C) on all software outputs, then update the hardware output with the resultant value. To combine multiple channels, a virtual output must be created for each voice, rather than updating the hardware output directly. These virtual outputs are never directly sonified but are combined; it is the resultant that is applied to the hardware output, creating the final waveform. Listing 3 demonstrates how this might be effectuated in C. Although two channels have been used in this example, there is theoretically no limit to the number of software channels that can be combined using PPM; one can continue append additional outputs with the bitwise OR operation. With each successive channel, one must employ progressively thinner widths to mitigate the increased probability of pulse collisions.

```
// process voice #1
if(--pitch_counter_1 == 0)
    pitch_counter_1 = frequency_1;

else if(pitch_counter_1 <= waveform_1)
    software_output_1 = 1;

else if(pitch_counter_1 >= waveform_1)
    software_output_1 = 0;
```

```
// process voice #2
if(--pitch_counter_2 == 0)
    pitch_counter_2 = frequency_2;

else if(pitch_counter_2 <= waveform_2)
    software_output_2 = 1;

else if(pitch_counter_2 >= waveform_2)
    software_output_2 = 0;

// combine software outputs
hardware_output = software_output_1 | software_output_2;
```

LISTING 3. A demonstration of PPM mixing in C pseudo-code. Although all channels have been addressed individually in this example, the companion `1-bit-generator.c` program employs a `for` loop to iterate through software channels and update outputs. This makes it easier to add or remove additional channels without requiring additional code and variables. When transcribing this example for other platforms (for example, ZX Spectrum machine code), one will probably find the paradigm used above (explicit declaration of each channel's processing code) more efficacious.

Whilst PPM is an effective method of crowding numerous frequencies into a single 1-bit waveform, the strict requirements of the technique sacrifices timbral variation for polyphony. As shown previously, square waves—perceptually the loudest width possible with the "richest" bass component—cannot be employed without heavy distortion; a square wave will interfere for 50% of its signal high duration. When juxtaposed with analog combinations of square waves, music made via PPM will sound contrastingly thinner, quieter, and lacking timbral "substance." One potential beneficial consequence of this concession is enforced aesthetic cohesion. The technique's demand for narrow duty cycles means that the dilemma presented in Figure 9 (pulse width modulation between larger widths is perceived less as a change in volume but instead a change in timbre) is circumvented by the process; the PPM 1-bit routine sounds like a single, cohesive instrument.

The second method of achieving polyphony does not suffer from a narrow timbral palette; this method is known to the 1-bit community as the *pulse interleaving method* (PIM).[75] One can imagine the implementation of this technique as, essentially, simultaneity by rapid arpeggiation. PIM operates on a fascinating premise: rather than mixing signals logically to fit within amplitudinal limitation (as with PPM), the technique switches, or arpeggiates, between voices at very high frequencies, so that only one monophonic waveform is expressed at any moment in time. The rapidity of oscillation between channels needed to achieve convincing polyphony is not particularly fast, but a problem arises as a result of quickly moving between two states: toggling a waveform from high to low (or vice versa) creates a click, or pulse. As with any pulse wave, because alternation between voices must happen at constant periodicity, this click produces a pulse train and, consequently, an audible, pitched sonority

75. utz, "Tutorial: How to Write a 1-Bit Music Routine," 1-Bit Forum, July 2015, http://randomflux.info/1bit/viewtopic.php?id=21.

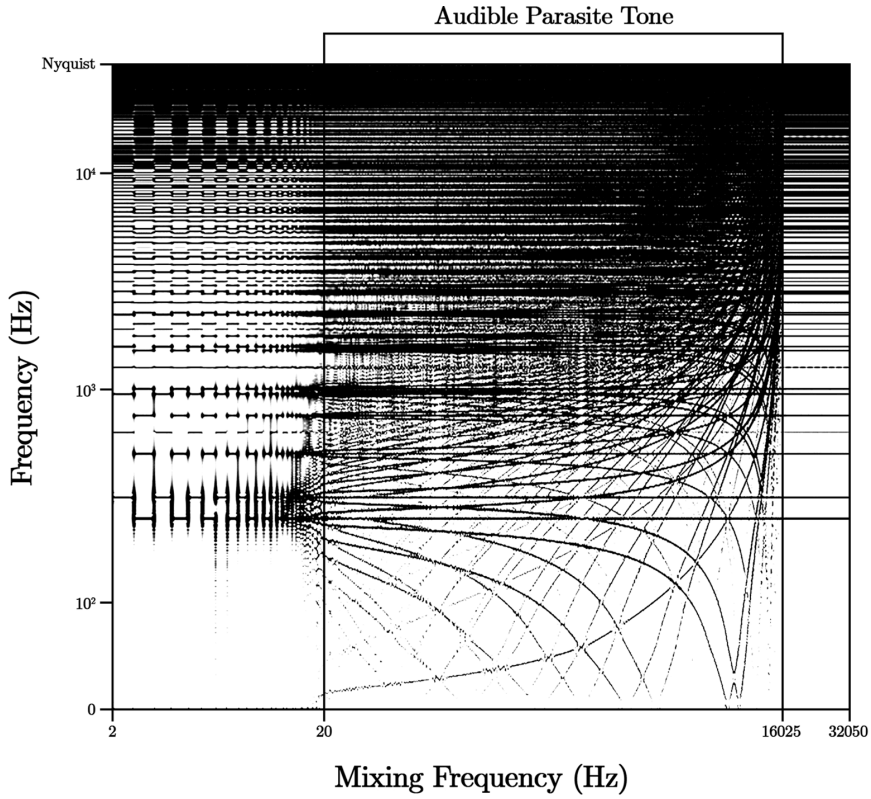

FIGURE 15. A spectrogram view of two 1-bit waveforms gradually combined using the pulse interleaving method. Pulses are mixed from 2 Hz through to 32050 Hz (Nyquist) to demonstrate how the parasite tone is generated then pushed upward, beyond audible frequencies. The visualization was created using Image-Line's Edison audio editor, and the example was generated at 64100 Hz by the `1-bit-generator.c` program. See pulse-interleaving-method.wav for an audio example of the unison and `pulse-interleaving-method-parasite-tone.wav` for the isolated parasite tone (the waveform generated by the mixing).

at the rate of interchange. To disguise this additional audible "parasite tone," the software mixing must happen either faster than human auditory perception or at a frequency beyond the limits of the medium replicating the signal—just like bandlimiting is used to alter amplitude (Figure 15). This familiar requirement suggests something intriguing: just as thin pulse widths produce changes in amplitude, as long as frequencies beyond the parasite tone are removed, the union can be considered true polyphony.

In the case of bandlimited PIM waveforms, when waveforms are merged they do not increase volume; rather, additional signals are progressively, *individually* quieter. This phenomenon can be explained by imagining how a speaker cone might move when a PIM signal is applied to it. A hypothetical diaphragm with a maximum response time (moving from stationary to fully extended) of f Hz is unable to react instantaneously to changes in voltage faster than this response time. If a signal's frequency exceeds f (let us assume the transmitting function is square) the diaphragm, after receiving a positive voltage, will move only a portion of its total distance before the signal falls to zero and the cone sympathetically

TABLE 1. A table showing the resultant, comparative volume of signals mixed via PIM. P_1 and P_2 are 1-bit waveforms of arbitrary pulse width (though very thin duty cycles will lower the perceptual volume of that voice, but not affect the ratio between the others), and the output represents the volume level in response to different input states.

P_1	P_2	Output
0	0	0
0	1	0.5
1	0	0.5
1	1	1

returns to rest. Therefore, if we send two concurrent 1-bit square waves to the speaker and oscillate between them faster than f, the diaphragm is unable to complete a full extension. There is, to the speaker, no parasite tone; instead we observe the following behavior, shown in Table 1.

The signals P_1 and P_2 are digital and can only exist in two states, 0 or 1. The output position of the diaphragm is assigned 0 at rest and 1 at maximum extension. The diaphragm will always attempt to act concordantly with the signal; however it can be "tricked" into generating a third state. This third position is an intermediate, caused by moving the speaker cone faster than it can respond, which leaves the diaphragm hovering at the average of the two voltages. We can see from Figure 16 that, when bandlimited in software, the behavior is identical to that of the thought experiment; a signal emerges representing three states.

The pulse interleaving method is not limited to two signals alone: depending on the frequency (the rate of oscillation between *all* virtual 1-bit signals must exceed the aforementioned f) more channels can be added with the trade-off that, with each channel, each individual signal's volume is ultimately quieter, and the computational demand increases, making it harder to reach f. The total volume of each subsequent channel is a subdivision of the maximum power equating to $v = \frac{1}{c}$, where v is the volume compared to the maximum and c is the total number of channels. The ratio between elements can be altered, but this will skew the relative loudness of outputs, as mixing priority will be assigned unequally. To achieve more complex ratios (not a simple $\frac{1}{n}$) requires higher resolutions where, rather than cycling equally between unique outputs, a singular channel is expressed for more samples than another.

There are two main approaches one might take to implement PIM in software. The first is to simply update the hardware output directly (in a similar fashion to Listings 1 and 2) but allow the mixing to be controlled by the program *control flow*. Listing 4 demonstrates how this might be achieved; the output is interleaved by the structure of the program: after the first channel is calculated and its value written to the output, the second is subsequently calculated and the current output is replaced with this new value. The program then loops around to replace the second channel's value with an updated value from the first, and so on. The caveat to this approach is that the amount of time

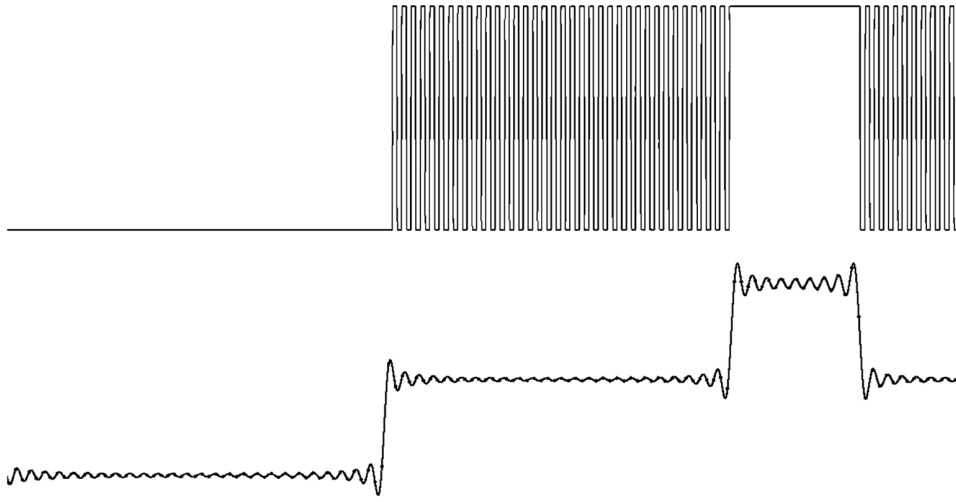

FIGURE 16. An oscilloscope view of a PIM waveform generated at 64100 Hz and, below, the same waveform downsampled to 44100 Hz.

taken to process the code between each output command must be identical to attain equivalent time spent expressing each channel's output value. If this is not achieved, the perceptual volume of the channels will be unbalanced. The second method is to manually control the interleaving by using an incrementing variable to switch between software outputs, shown in Listing 5. With this technique, the outputs are balanced because each time the code reaches an instruction to change the output, the channel to be expressed is switched.

```
// process voice #1
if(--pitch_counter_1 == 0)
    pitch_counter_1 = frequency_1;

else if(pitch_counter_1 <= waveform_1)
    output = 1;

else if(pitch_counter_1 >= waveform_1)
    output = 0;

// process voice #2
if(--pitch_counter_2 == 0)
    pitch_counter_2 = frequency_2;

else if(pitch_counter_2 <= waveform_2)
    output = 1;

else if(pitch_counter_2 >= waveform_2)
    output = 0;
```

LISTING 4. A demonstration of PIM mixing in C pseudo-code, interleaved via the program control flow. The interleaving occurs at the speed the program can process each command.

```
// process voice #1
if(--pitch_counter_1 == 0)
    pitch_counter_1 = frequency_1;

else if(pitch_counter_1 <= waveform_1)
    software_output_1 = 1;

else if(pitch_counter_1 >= waveform_1)
    software_output_1 = 0;

// process voice #2
if(--pitch_counter_2 == 0)
    pitch_counter_2 = frequency_2;

else if(pitch_counter_2 <= waveform_2)
    software_output_2 = 1;

else if(pitch_counter_2 >= waveform_2)
    software_output_2 = 0;

// switch software outputs
if(current_output == 0){
    hardware_output = software_output_1;
    ++current_output;
}
else if(current_output == 1){
    hardware_output = software_output_2;
    current_output = 0;
}
```

LISTING 5. A demonstration of PIM mixing in C pseudo-code, interleaved via a switching variable. One might also use software outputs and place `hardware_output` update commands throughout the code, ensuring that the timing is equal between each update; it is certainly a visually messy solution but works effectively in practice on embedded systems.

PIM is actually a form of pulse width modulation but performed so rapidly that the previously described timbral characteristics are imperceptible. Using a pulse width–modulated pulse train in this way, it is possible to generate continuous amplitudinal variation using a digital signal alone, thus any number of voices can be summed (i.e., replication of any combination of instruments or existing recording is possible). Pulse width modulation describes the method by which a succession of pulses are delivered at certain frequencies to regulate the voltage of a signal. When this signal is applied to a speaker or similar output device, the speaker acts as if it were receiving an analog signal. The emergent signal is a product of the average value of the waveform, when modulated on a fixed frequency, and appears, to the system receiving the signal, to be a true analog signal. The pulse interleaving method can therefore be considered as a low-resolution PWM. PIM is *much* simpler to implement in software, however, as the bit depth need only be the desired number of voices. PIM employs each "bit" of PWM resolution to convey an individual 1-bit channel, capable of independent duty cycles.

It should be added that I do not really consider high-resolution PWM *alone* an *aesthetic* 1-bit solution; it is distinct in that the goal of PWM as a technique is to be *imperceptible*, existing only as the signal it is attempting to recreate. Although there are impressive examples of PWM samplers implemented in 1-bit audio routines (some contemporary 1-bit routines rely solely on this method, such as utz's *stringks* engine[76]) these are always (technically) *poor* implementations, in that they produce distinguishing artifacts—the process is audible to the listener. This attitude reflects one seen in the ethos of wider chipmusic practice, where platform authenticity is contextually important to the artifact.[77] This may inform how the composer will wish to design their software routine.

The seeming process duality of PPM versus PIM is reliant on two assumptions: first that the composer wishes to use polyphony at all (which was not always necessary—for example, *Rockman* on the ZX Spectrum[78] and the DOS version of *The Secret of Monkey Island*[79] both employ monophonic arpeggiation between "simultaneous" voices to imply polyphony, rather than PIM/PPM techniques), and second that the composer is using embedded, or legacy, architectures (such as the ZX Spectrum or early IBM PC). The concessions and limitations described above are considerations one must make when there is a technical restriction placed upon the music. If the composer wishes to merely evoke 1-bit instrumental aesthetics, then one might opt for the most idiosyncratic approach—or that which is most recognizably 1-bit. Brian Eno's observation that the signature imperfections of a medium become its defining characteristics is certainly apt in this situation.[80] I would personally argue that *the* instrumental 1-bit idiolect, irrespective of chosen platform, is PPM. Although those 1-bit routines that employ PIM are often imperfect in practice, when executed by a system capable of perfectly executing commands at the correct frequency, PIM is indistinguishable from true polyphony.

There are *many* more techniques I have not explored in this article. For example, use of pitch for percussive effect, granular synthesis, frequency modulation, and further approaches not documented here—or perhaps anywhere. There is certainly still room for innovation; contemporary 1-bit practice has had a brief period of time to develop and explore techniques, compared to other instruments such as the piano. Contemporary 1-bit sonics were only commercially relevant for perhaps a decade before relegation to obsolescence and relative obscurity. It seems a shame to abandon what is a unique aesthetic environment with musically interesting instrumental capabilities, presumably in the pursuit of increased realism. This said, contemporary routines are still being developed and can be far more sophisticated than legacy implementations; some boast 15 concurrent voices and others include sample playback.[81] The 1-bit scene is small but has an enthusiastic, active community

76. utz, "Utz82/Zx-Spectrum-1-Bit-Routines," GitHub, 2018, https://github.com/utz82/ZX-Spectrum-1-Bit-Routines/tree/master/stringks.

77. Sebastian Tomczak, "Authenticity and Emulation: Chiptune in the Early Twenty-First Century," Conference Paper at the International Computer Music Conference, August 2008, https://quod.lib.umich.edu/cgi/p/pod/dod-idx/authenticity-and-emulation-chiptune-in-the-early-twenty.pdf?c=icmc;idno=bbp2372.2008.035.

78. *Rockman*, 1987, Dene T. Carter/Mastertronic Ltd, ZX Spectrum.

79. *The Secret of Monkey Island,* 1990, LucasFilm Games/LucasArts Entertainment, MS-DOS.

80. Brian Eno, *A Year with Swollen Appendices* (London: Faber and Faber, 1996).

81. utz, "Sound Routines," Irrlicht Project – Code, http://irrlichtproject.de/code.php.

with new releases for ZX Spectrum,[82] PC Speaker,[83] TI series Texas Instruments Graphing Calculator,[84] and others. It is an understatement to say that I have learned a lot from the philanthropic efforts of online 1-bit enthusiasts such as utz and Alex "shiru" Semenov, both of whom have performed crucial roles in the documentation and propagation of 1-bit music as well as being talented musicians and coders. I certainly hope that this investigation will further their efforts in the aggregation, engagement, and expansion of the existing literature, as well as encourage the reader to try implementing these ideas in their own routines and create some exciting new sounds.

BIBLIOGRAPHY

"Ada Lovelace." The Babbage Engine. Computer History Museum. http://www.computerhistory.org/babbage/adalovelace/.

"First Digital Music Made in Manchester." Technology. The University of Manchester, June 2008. http://www.manchester.ac.uk/discover/news/first-digital-music-made-in-manchester.

"Interview with David Wise (December 2010)." Square Enix Music Online. 2010. https://www.squareenixmusic.com/features/interviews/davidwise.shtml.

"Nellie: School Computer." Tomorrow's World, Series 4. BBC Broadcasting Service. February 1969. http://www.bbc.co.uk/programmes/p0154hns.

"Phase." National Institute of Standards and Technology. September 2016. https://www.nist.gov/time-and-frequency-services/p.

"Why Not Always Cut the 20–30 Hz Range?" KVR Audio. https://www.kvraudio.com/forum/viewtopic.php?f=62&t=313807&sid=7da7241355268614d8690cad3702890b.

Access Software Inc. *Crime Wave Instruction Manual*. Birmingham, UK: US. Gold Ltd, 1990.

Allen, Joe. "How to Make 8-Bit Music: An Introduction to Famitracker." Synthtopia. May 2015. http://www.synthtopia.com/content/2015/05/01/how-to-make-8-bit-music-an-introduction-to-famitracker/.

Ball, Phillip. *The Music Instinct*. London: The Bodley Head, 2010.

Bland, Barb. "Making Complex Waves." 1999. http://hep.physics.indiana.edu/~rickv/Making_complex_waves.html

blargg. "NES Apu Sound Hardware Reference." nesdev.com, 2004. http://nesdev.com/apu_ref.txt.

Borwick, John. "Frequency." In *The Oxford Companion to Music*, edited by Alison Latham. Oxford, UK: Oxford University Press, 2011. http://www.oxfordmusiconline.com/subscriber/article/opr/t114/e2693

Brink. "M'Lady." ZX Art. 2014. https://zxart.ee/eng/authors/b/johan-elebrink/mlady/.

Butler, Lloyd. "Waveforms Using the Cathode Ray Oscilloscope." Waveform and Spectrum Analysis. June 1989. http://users.tpg.com.au/users/ldbutler/Waveforms.htm

Carlsson, Anders. "TIMELINE." ChipFlip, 2017. https://chipflip.wordpress.com/timeline/.

Chronos. 1987. Mastertronic Ltd. ZX Spectrum.

Copeland, Jack, and Jason Long. "Christmas Carols from Turing's Computer." *Sound and Vision Blog*. 2017. https://blogs.bl.uk/sound-and-vision/2017/12/christmas-carols-from-turings-computer.html.

Copeland, Jack, and Jason Long. "Restoring the First Recording of Computer Music." *Sound and Vision Blog*. 2016. https://blogs.bl.uk/sound-and-vision/2016/09/restoring-the-first-recording-of-computer-music.html.

82. Irrlicht Project, "Dat Fuzz," March 2014, https://irrlichtproject.bandcamp.com/releases.

83. shiru, "System Beeps," January 2019, https://shiru8bit.bandcamp.com/album/system-beeps.

84. utz, "Utz82/Zx-Spectrum-1-Bit-Routines," GitHub, 2018, https://github.com/utz82/ZX-Spectrum-1-Bit-Routines/tree/master/stringks.

Corey, J. *Audio Production and Critical Listening: Technical Ear Training*. Abingdon, UK: Taylor & Francis, 2016.
Crime Wave. 1990. Access Software Inc. MS-DOS.
Davis, Gary, Ralph Jones, and Y.I. Corporation. *The Sound Reinforcement Handbook*. Milwaukee, WI: Hal Leonard, 1989.
Deák, Ján. "ZX-3." World of Spectrum. 1991. http://www.worldofspectrum.org/infoseekid.cgi?id=0027576.
Deák, Ján. "ZX-7." World of Spectrum. 1990. http://www.worldofspectrum.org/infoseekid.cgi?id=0012538.
Dambrin, Didier. "3x Osc." *FL Studio 20 Reference Manual*. n.d. http://www.image-line.com/support/flstudio_online_manual/html/plugins/3x%20OSC.htm.
Doornbusch, Paul. "Computer Sound Synthesis in 1951: The Music of CSIRAC." *Computer Music Journal* 28 (2004): 10–25. https://doi.org/10.1162/014892604322970616.
DOOM. 1993. id Software. MS-DOS.
Driscoll, Kevin, and Joshua Diaz. "Endless Loop: A Brief History of Chiptunes." *Transformative Works and Cultures 2*. 2009.
Eno, Brian. *A Year with Swollen Appendices*. London: Faber and Faber, 1996.
Everbach, Erich Carr. "Noise Quantification and Monitoring: An Overview." The Science Building Project. 2000. http://www.swarthmore.edu/NatSci/sciproject/noise/noisequant.html
Falstad, Paul. Fourier Series Applet. http://www.falstad.com/fourier/.
Fildes, Jonathan. "'Oldest' Computer Music Unveiled." Technology. BBC News. June 2008. http://news.bbc.co.uk/1/hi/technology/7458479.stm.
Follin, Tim. "Star Tip 2." *Your Sinclair* no. 20 (1987).
Fritsch, Melanie. "History of Video Game Music." In *Music and Game: Perspectives on a Popular Alliance,* edited by Peter Moormann. Wiesbaden, Germany: Springer, 2012. https://doi.org/10.1007/978-3-531-18913-0_1.
Haber, Howard E. "How to Add Sine Functions of Different Amplitude and Phase." 2009. http://scipp.ucsc.edu/~haber/ph5B/addsine.pdf.
Hardy, Norman. "Music." Stories. 2005. http://www.cap-lore.com/stories/music.html.
Hartley, David. "EDSAC 1 and After: A Compilation of Personal Reminiscences." EDSAC 99 Conference, 1999. https://www.cl.cam.ac.uk/events/EDSAC99/reminiscences/.
Herman, Gary. *Micromusic for the Commodore 64 and BBC Computer*. London: PAPERMAC, 1985.
Hopkins, Christopher. "Chiptune Music: An Exploration of Compositional Techniques Found in Sunsoft Games for the Nintendo Entertainment System and Famicom from 1988–1992." PhD diss., Five Towns College. 2015.
Image-Line. "Edison." *FL Studio 20 Reference Manual*. n.d. https://www.image-line.com/support/flstudio_online_manual/html/plugins/Edison.htm.
irrlicht project. "Dat Fuzz." March 2014. https://irrlichtproject.bandcamp.com/releases.
Karlsson, Magnus. "In memoriam" DATASAAB, 1999.https://www.ctrl-c.liu.se/misc/datasaab/index-eng.html
Kleiman, Kathryn. "Singing Binac - 1948." CYHIST Community Memory: Discussion List on the History of Cyberspace. November 1997. https://groups.yahoo.com/neo/groups/cyhist/conversations/messages/1271.
Lakawicz, Steve. "The Difference Between Pulse Waves and Square Waves." Research in Game Music. *Classical Gaming*. May 2012. https://classicalgaming.wordpress.com/2012/05/15/research-in-game-music-the-difference-between-pulse-waves-and-square-waves/.
Latham, Alison, ed. "White Noise." In *The Oxford Companion to Music*. Oxford, UK: Oxford University Press, 2011. http://www.oxfordmusiconline.com/subscriber/article/opr/t114/e8240.

Latimer, Joey. "Hit It, Maestro!" *Compute! Magazine*, 1990. https://web.archive.org/web/20140906061745/http://www.joeylatimer.com/pdf/Compute!%20April%201990%20PC%20Sound%20Gets%20Serious%20by%20Joey%20Latimer.pdf.

Levitin, Daniel J. *This Is Your Brain on Music: The Science of a Human Obsession*. New York: Dutton, 2006.

Louis, E. G. "Practical Techniques of Square-Wave Testing." Radio & TV News. RF Cafe. July 1957. http://www.rfcafe.com/references/radio-news/practical-techniques-square-wave-testing-july-1957-radio-tv-news.htm.

Lynch, Gerald. "From 8-Bit to Chiptune: The Music That Changed Gaming Forever." Techradar, March 2017. http://www.techradar.com/news/8-bit-music-the-soundtrack-to-a-gaming-revolution-that-resonates-today.

Marshall, Brian. "Last Ninja 2." 1988. ZX Spectrum. System 3 Software Ltd.

Middleton, R. G. *Know Your Square Wave and Pulse Generators*. Carmel, IN: H. W. Sams, 1965.

Nastase, Adrian S. "How to Derive the RMS Value of Pulse and Square Waveforms." Mastering ElectronicsDesign.com. 2012. https://masteringelectronicsdesign.com/how-to-derive-the-rms-value-of-pulse-and-square-waveforms/.

Mister BEEP. "Chromospheric Flares." ZX Art. 2014. https://zxart.ee/eng/authors/m/mister-beep/chromospheric-flares/.

Nave, Carl Rod. "Clarinet Waveform." 1998. http://hyperphysics.phy-astr.gsu.edu/hbase/Music/clarw.html.

Ocean Loader 3. 1987. Peter Clarke/Ocean Software. Commodore 64 Loading Software.

Ögren, Joakim. "The Hardware Book." *Compute! Magazine*. 1997. http://www.acc.umu.se/~stric/tmp/hwb13pdf/hwbook.pdf#page=290

Ohanesian, Liz. "What, Exactly, Is 8-Bit Music?" *LA Weekly*. August 2011. http://www.laweekly.com/music/what-exactly-is-8-bit-music-2409754

Oldham, Guy. "Harmonics." In *The Oxford Companion to Music*, edited by Alison Latham. Oxford, UK: "Harmonics." In *The Oxford Companion to Music*, edited by Oxford University Press, 2011. http://www.oxfordmusiconline.com/subscriber/article/grove/music/50023.

Orfeus Music Assembler. 1990. Proxima Software. ZX Spectrum.

Paul. "Chronos." *Crash* 41 (1987). https://archive.org/details/crash-magazine-41.

Phelps, Philip. 2007. "A Modern Implementation of Chiptune Synthesis." University of the West of England. 2007. https://woolyss.com/chipmusic/chipmusic-discovery/PhillPhelps-ChiptuneSynth.pdf.

Pohlmann, Ken C. *Principles of Digital Audio*, 6th ed. New York: McGraw-Hill, 2011.

Prestini, Elena, *The Evolution of Applied Harmonic Analysis: Models of the Real World.* Boston: Birkhäuser, 2004.

Prince of Persia. 1989. Brøderbund. MS-DOS.

Raphaelgoulart. "Surprisingly NOT Four Twenty." ZX Art. 2014. https://zxart.ee/eng/authors/r/raphaelgoulart/surprisingly-not-four-twenty/.

Rockman. 1987. Dene T. Carter/Mastertronic Ltd. ZX Spectrum.

Rood, Sarah. *From Visions to Realities*. Melbourne: Monash University Custom Publishing Services, 2008.

Russell, Daniel A. "Acoustics and Vibration Animations." The Pennsylvania State University, 2014. https://www.acs.psu.edu/drussell/demos/superposition/superposition.html.

Rutherford-Johnson, Tim, ed. "Envelope" In *Oxford Dictionary of Music*, 6th ed. Oxford, UK: Oxford University Press, 2013.

shiru. "System Beeps." January 2019. https://shiru8bit.bandcamp.com/album/system-beeps.

Silvast, Antti, Markku Reunanen, and Gleb Albert. "Demoscene Research." http://www.kameli.net/demoresearch2/.

SimCity 2000. 1993. Maxis. MS-DOS.
Smith, Steven. *The Scientist and Engineer's Guide to Digital Signal Processing*. San Diego: California Technical Publishing, 1997.
Software Technology Corporation. *Music System User's Manual*. 1977. http://www.sol20.org/manuals/music.pdf.
Sordillo, David. "Music Playing on the Pdp-6." Project MAC. 1966. ftp://publications.ai.mit.edu/ai-publications/pdf/AIM-107.pdf.
Strange, David. "ADSR Envelope Generator." *Electronic Soundmaker,* no. 2 (1984).
Sundqvist, Göran. D21 - In Memoriam - D22. (Vinyl, EP) Sweden: KDA, 2019.
Sweetwater. "Pink Noise Versus White Noise." *inSync*. 2000. https://www.sweetwater.com/insync/pink-noise-versus-white-noise/.
The Secret of Monkey Island. 1990. LucasFilm Games/LucasArts Entertainment. MS-DOS.
The Sentinel. 1987. Software Creations/Firebird Software Ltd. ZX Spectrum.
Todd, Patrick "Bucky," and Steve "Ap0c" Lakawicz. "Interview with David Warhol (Composer, Programmer)." Video Game History. December 2016. http://www.vgarc.org/vgarc-originals/interview-with-david-warhol/.
Tomczak, Sebastian. 2008. "Authenticity and Emulation: Chiptune in the Early Twenty-First Century." Conference Paper at the International Computer Music Conference. August 2008. https://quod.lib.umich.edu/cgi/p/pod/dod-idx/authenticity-and-emulation-chiptune-in-the-early-twenty.pdf?c=icmc;idno=bbp2372.2008.035.
Total Eclipse. 1988. Incentive Software. MS-DOS.
Turing, Alan. *Programmers' Handbook for Manchester Electronic Computer Mark II*. The Manchester Computer. http://www.alanturing.net/turing_archive/archive/m/m01/M01-030.html.
utz. "Computer Music in 1949?" Ancient Wonderland. Irrlicht Project. November 2015. http://irrlichtproject.blogspot.co.uk/2015/11/computer-music-in-1949.html.
utz. "HoustonTracker 2." March 2018. http://www.irrlichtproject.de/houston/.
utz. "Sound Routines." Irrlicht Project - Code. http://irrlichtproject.de/code.php
utz. "Topic: A Timeline of 1-Bit Music (1949–1979)." 1-Bit Forum. 2015. http://randomflux.info/1bit/viewtopic.php?id=40.
utz. "Tutorial: How to Write a 1-Bit Music Routine." 1-Bit Forum. July 2015. http://randomflux.info/1bit/viewtopic.php?id=21.
utz. "Utz82/Zx-Spectrum-1-Bit-Routines." *GitHub*. 2018. https://github.com/utz82/ZX-Spectrum-1-Bit-Routines/tree/master/stringks.
Van Keuren, Thomas. "Ebb Tide Played by 1970 Univac Computer (No Sound Card)." January 2015. https://www.youtube.com/watch?v=X6F7qwa5TZg
Vickers, Steven. *ZX Spectrum Basic Programming*. Edited by R. Bradbeer. Cambridge, UK: Sinclair Research, 1982.
Weixelbaum, Herbert. "Game Boy Sound Comparison." *Game Boy Music*. June 2019. http://www.herbertweixelbaum.com/comparison.htm.
YERZMYEY. "1-Bit Chiptunes/Beeper Music." *Chipmusic*. 2013. http://chipmusic.org/forums/topic/12566/1bit-chiptunes-beeper-music/.
Zuse. "Programm 47." *Zuse Z23 Programmkatalog*. 1970. http://www.sol20.org/manuals/music.pdf.
ZX-3. 1991. Ján Deák/Elektronika. ZX Spectrum.