On the Development of an Interface Framework in Chipmusic: Theoretical Context, Case Studies and Creative Outcomes

by

Sebastian Tomczak

Submitted in fulfilment of the requirements for the degree of

Doctor of Philosophy

Elder Conservatorium of Music
Faculty of Humanities and Social Sciences
The University of Adelaide

March, 2011

Table of Contents

List of Figures and Tables

Abstract

The current study deals with the development, application and outcomes of an Interface Framework that can be applied to a range of video game systems in order to create real-time music. The Framework controls the internal sound chips of video game consoles from a host computer or external device in such a way as to greatly expand the artistic and technical choices available to composers and musicians. The submission comprises a written component, a data component (made up of timelines, source code and schematics) and a creative outcomes component (made up of approximately one hour of music).

Declaration

Sebastian Tomczak Date:

Acknowledgements

This submission would not have been possible without the assistance, guidance and support from colleagues, friends, and family.

I am in great debt for the support, criticism and patience extended to me by my supervisory panel – Dr Mark Carroll and Christian Haines. I acknowledge the role that Dr Carroll has had in editing the written portion of the submission, in which he has helped deal with issues of language, expression and consistency. Mr Haines has been a concrete support for technical concepts, issues of written expression and style, and for the development and implementation of ideas.

Additionally, I wish to thank my peers and colleagues – Tristan Louth-Robins, Luke Harrald, Hayley Miller, Peter Sansom and Poppi Doser – as well as the two inspirational music technologists that I have had the pleasure of working / performing with throughout the course of this study – Nicolas Collins and Paul Slocum.

Finally, this submission could not have taken place without the support of my wife, Lauren, and my parents, Chris and Matt. You have my gratitude.

Introduction: Issues and Perspectives

With a history spanning over twenty-five years, chipmusic has become a global phenomenon. The term chipmusic here refers to music composed or produced using the technological processes or aesthetic influences associated with obsolete computers and video game consoles. It has reached a wide audience through regular events such as Blip Festival, Pulse Wave and Micro Disco, as well as online resources and communities. Elements of chipmusic have flowed into popular musical forms such as hip-hop, pop and rock music. The importance and relevance of chipmusic reached a highpoint in the late twentieth and early twenty-first centuries, when developers and composers began applying evolving computer technologies in order to make fresh use of selected obsolete video game consoles as modern-day creative tools and outlets for musical expression. To an extent this has led to a situation which has seen chipmusic practitioners favour a narrow range of gaming consoles and tools with which to ply their trade. The current study seeks to break that impasse by offering an Interface Framework that is applicable to a wider range of game consoles, one that reduces the process of developing real-time musical control applications to a series of discrete, easily implemented steps. The study documents the development of the Framework and provides the schematics and software codes produced by it. A series of creative works attest to the efficacy of the Framework and its value to the chipmusic community as a whole.

In a wider historical context, the ideas underpinning chipmusic relate to a number of other branches within music and music technology, as well as works and projects previously undertaken by various composers. This includes the Futurist movement, with Luigi Russolo's noise intonators and the links to industrial sounds[1], the technological innovation of Theremin, Cowell and Schillinger with the first drum machine (the

1 Russolo, Luigi. 'The Art of Noises'. 1913. Letter to Francesco Balila Pratella.
2 Hopkin, Ben. 'American Mavericks: The Rhythmicon'. http://musicmavericks.publicradio.org/rhythmicon/ (Accessed 22 February 2011).

Rhythmicon[2]) and the recontextualisation of the turntable by composers such as Wolpe, Kurt Weill and Hindemith in the 1920's[3] and later by John Cage[4]. The 1960's saw the birth of circuit bending by Reed Ghazala, a musically unique and creative approach to technological appropriation and exploitation[5]. The period from the 1960's to the 1980's saw a number of music technologists and composers building their own electronic instruments, thus making the most of the (then) newly available integrated circuits. This includes works by David Behrman, Salvator Martirano, Stanley Lunetta, Greg Schiemer, Rainer Linz, Carl Vine and Warren Burt. More closely linked to chipmusic is work in the 1980's making use of microprocessors by exploiting the unique characteristics of standalone sound chips. Composers and groups included George Lewis, the League of Automatic Music Composers and certain projects of the STEIM[6] group as well as Schiemer, Linz and Burt.

Modern chipmusic is directly linked to what is termed the 'demoscene'. The demoscene, having evolved in the early 1980s, is a subculture that centres on the programming of non-interactive demonstrations that explore and exploit the shortcomings inherent in early home computers and video game systems.[7] Conceptually speaking, the demoscene in its early years was concerned more with the technical journey than the creative destination, with negotiating the technical limitations of the sound hardware

2 Hopkin, Ben. 'American Mavericks: The Rhythmicon'. http://musicmavericks.publicradio.org/rhythmicon/ (Accessed 22 February 2011).
3 Ross, Alex. 'The Record Effect', in The Best of Technology Writing 2006' (Ann Arbor: The University of Michigan Press, 2006), 35.
4 Cage, John. 'Cartridge Music'. Music composition. 1960. Premiere: October 6, 1960, Cologne, Germany,
5 Ghazala, Reed. 'Exploring Circuit-Bending Today', in *Circuit Bending: Build Your Own Alien Instruments* (Indianapolis: Wiley Publishing, 2005), 1 – 25.
6 'STEIM'. http://www.steim.org/steim/about.html. (Accessed 22 February 2011).
7 Markuu Reunanen and Antti Silvast. 'Demoscene Platforms: A Case Study on the Adoption of Home Computers', in *History of Nordic Computing 2*, J. Impagliazzo, T. Järvi, and P. Paju, eds. (New York: Springer Publishing, 2009) 289 – 301.

embedded in those systems rather than focussing on the creative results.[8] Demoscene composers and programmers nevertheless employed archetypes and elements that have continued to remain popular in the modern era of chipmusic. Examples include the sound of a snare drum, a chorus-like lead line, or the shape and speed of an arpeggio that is used to simulate chords in a progression.

The mid to late 1990s saw the splintering of chipmusic from the demoscene. An important contributing factor to the split was emergence of affordable tools, in the form of both hardware and software, which allowed certain computers and video game systems to be easily used for the composition of music. Developers gained access to hardware, such as flash carts for the Nintendo Game Boy, which facilitated the creation of music composition software that circumvented many of the processes with which the demoscene was preoccupied. The emergence of tracker music packages for personal computers allowed musicians to create sample-based and synthesis-based chip music, such as one finds in the sample-based Fasttracker 2.[9]

In recent years, the now-superseded Nintendo Game Boy has become synonymous with chipmusic. This is partly due to the fact that two main programs for the system, Nanoloop[10] and LSDJ[11], are still being updated by their authors today. Such is the popularity of the Nintendo Game Boy that it appears to have drawn many composers and performers from outside the demoscene to chipmusic. The attraction lies in its thoroughly documented, well-supported music software, and its portability and availability in

8 Kevin Driscoll and Joshua Diaz. 'Endless Loop: A Brief History of Chiptunes', in *TransformativeWorks and Cultures*, vol. 2 (2009). http://journal.transformativeworks.org/index.php/twc/article/view/96/94 (accessed 15 March 2010).

9 Fredrik Huss and Magnus Högdahl. Fast Tracker 2. Software for DOS. http://www.gwinternet.com/music/ft2/software/ft2.htm (accessed 10 March 2010).

10 Oliver Wittchow. Nanoloop. Software for Game Boy. http://nanoloop.com/node/13 (accessed 15 March 2010).

11 Johan Kotlinski. Little Sound DJ. Software for Game Boy. http://www.littlesounddj.com/lsd/ (accessed 15 March 2010).

comparison to other obsolete video game systems. Additionally, pioneering musical works by Game Boy musicians such as Nullsleep and Bit-Shifter have led to an increased awareness of Game Boy music in the early 21st century.

Chipmusic can be considered under two umbrella concepts: chipmusic as a process-specific phenomenon and chipmusic as a genre-specific phenomenon. The first overarching concept – chipmusic as process – relates to the composition of music via the utilisation of obsolete computer and video game hardware. This includes a number of processes and technical abstractions. One such process is the use of the integrated sound-making mechanisms found within obsolete computers and video game systems. This includes the use of separate, standalone hardware that stems or is salvaged from these gaming systems, as well as software that attempts to emulate the principles upon which these mechanisms are based. As a generalisation, this idea can be understood as referring to the medium, processes and techniques involved from a technical and sonic perspective in creating an expressive musical outcome. This umbrella term also covers the use of specific data formats for the composition, playback, storage and archiving of music. In this regard, the process is not informed by the musical outcome.

The second overarching concept informing current trends is the idea of chipmusic as a genre-specific phenomenon. This relates to composition utilising certain stylistic aspects of music, in particular selective, stylistic mimicry of computer and video game sound tracks from the eight and sixteen-bit eras (early 1980s to the mid 1990s). The compositional result covers specific techniques, timbral variation, key signatures, time signatures, tempi and other aspects of musical style. This implies a divorce from the constraints of specific processes to achieve a given aesthetic or musical outcome. 'Chipmusic as genre' covers music that has been composed as if influenced by, related to or part of an obsolete video game, without regard as to how the sound has been produced.

For example, a square wave oscillator that is not related to a video game sound chip might be used. However, the musical outcome might have a similar aesthetic as music that is played on specific sound chips. The musical outcome is not informed by the process.

It is important to differentiate between the idea of chipmusic as process versus genre, as the identity of chipmusic culture and musical practise relies heavily on specialised technologies as well as the aesthetic of video game music. This ties in to the fact that, as with modern electronic music in general, the composition of chipmusic is strongly connected to certain production techniques and processes, instrumentation, human-machine interfacing and timbral outcomes. These elements inform and guide the composition process.

These two concepts can be thought of as feeding into and ultimately helping to form a coherent topology of what defines chipmusic in the modern era. Demoscene scholars Kevin Driscoll and Joshua Diaz summarise the combination of chipmusic as process and as genre in the modern era as follows:

> Born out of technical limitation, their soaring flutelike melodies, buzzing square wave bass, rapid arpeggios, and noisy gated percussion eventually came to define a style of its own, which is being called forth by today's pop composers as a matter of preference rather than necessity.[12]

The chipmusic scholar and historian Anders Carlsson subsumes these concepts under the umbrella of chipmusic as culture:

> Traditionally, chipmusic has been defined by what gadgets were used to make the music. As the sounds of 8-bit machines has influenced and been influenced by pop culture at large, the technological definition proved to be problematic for general use. A musical genre definition developed during the 21st century. For some this leads to a loss of authenticity, while others welcomed it as a loss of elitist techno-fundamentalism. I

12 Kevin Driscoll and Joshua Diaz. 'Endless Loop: A Brief History of Chiptunes'.

> distinguish between chipmusic as medium and chipmusic as form... Chipmusic can also be analyzed as a subculture with its own communication media, norms, status makers, artifacts, and software.[13]

The concept and definition of chipmusic as a (sub)cultural phenomenon is important to the overall interpretation and analysis in the context of modern and historical music practise. However, due to the nature of this work, the first two definitions (as process and as genre) will play a larger role in relation to the creative, technical and theoretical outcomes of the current study.

The two processes at the heart of chipmusic are hardware-based synthesis and emulation. Hardware-based synthesis is the process of generating audio waveforms using integrated circuits in the form of sound chips that are, in this instance, located within obsolete game consoles. Emulation is concerned with digitally recreating the functionality of a sound chip, whether that be the actual sound generated or any potential control properties the chip may possess. The latter are understood here as being, for example, integrated amplitude modulation that is executed completely within the sound chip, as opposed to being governed by an external source such as a microcomputer. Emulation plays a major role in modern chipmusic as it combines the ease of use afforded by the contemporary electronic music studio with distinct timbres generated by sound chips extracted from obsolete technology. An excellent example of a recent chipmusic emulation focussing on tone generation is the Chipsounds plugin, developed by David Viens of Plogue.[14]

Regardless of the benefits associated with emulation, hardware-based synthesis continues to play an intrinsic and integral role in modern chipmusic. Although sound chips

13 Anders Carlsson. 'Chipmusic', in *Chipflip*. http://chipflip.wordpress.com/chipmusic/ (accessed 16 March 2010).

14 David Viens. Chipsounds. 2009. Software for Mac OS X and Windows. Published by Plogue. http://www.plogue.com/?page_id=43 (accessed 15 March 2010).

can be emulated so that the sonic outcome is almost identical to the audio output of the chip, the virtualisation often falls short. This is the case particularly with regard to frequency response, signal to noise ratio and inherent noise bands. Indeed, it is the supposed sonic 'shortcomings' of physical systems that give them their appeal. Additionally, emulation is less concerned with controlling properties that are extrinsic to the functionality of the chip itself. It is those external control properties – which may include amplitude, frequency and timbral modulation – that in fact maximise the capabilities and sonic outcomes that can be derived from a given sound chip. When taking into account most video game systems, variations in electronic circuitry on a model-by-model and unit-by-unit basis are also difficult to emulate. The current study is not concerned with emulation, but with the development and use of a number of hardware-based video game and sound chip devices. In effect, the Framework offered here allows the user to target sound chips in situ, and then effectively force those same chips to perform musical functions beyond what would normally be expected of them within the context of producing a video game sound track.

As a generalisation, the hardware employed by chipmusicians is drawn from the late 1970s to the mid 1990s.Why are obsolete devices such as the Nintendo Entertainment System, the Game Boy, the SEGA Mega Drive or the Commodore 64 computer so valued from a creative perspective? The answer lies in part with the method in which modern video game and computer systems handle sound playback in comparison to their obsolete counterparts. Modern systems feature homogeneous audio hardware that by and large does not colour or affect the sound.[15] Sound playback consists of complex waveforms that are either synthesised in real-time or played back from a sample library. The current standard

15 Aaron Marks. 'An Introduction to Game Audio', in *The Complete Guide to Game Audio: For Composers, Musicians, Sound Designers, Game Developers* (Burlington: Focal Press, 2008), 1 - 21.

for audio within gaming is of CD quality (with a bit depth of 16 bits and a sample rate of 44.1kHz) or a comparable lossy or lossless compressed audio format. Audio material may be mixed for stereo or multi-channel spatialisation.

By contrast, each video game console of the vintage era has its own characteristic sound palette and audio capabilities, resulting in signature timbres, instrumentation and tuning. Over time these facets have helped to create a certain overall character for a given video game or computer system. Examples include the Commodore 64's pulsewave with its high-resolution duty cycle, the Nintendo Entertainment System's low-frequency triangle bass channel or the unique timbres and limited pitch set tuning of the Atari 2600. Sound playback hardware such as this is overly specialised due to constraints of memory, computational power and manufacturing cost. These factors are no longer issues in modern systems. This overspecialisation of real-time synthesis and sample playback hardware has resulted in peculiarities, limitations and interesting features on a system-by-system basis, in contrast to the modern homogeneity of video game and computer audio playback.

This study is inspired by the subversive application of technology for the purposes of musical expression, production and performance undertaken by pioneers such as Nicolas Collins and Reed Ghazala, as well as creative inventors and developers such as Gijs Gieskes, Chris Kann and Paul Slocum. The aspects, ideas and creative results expressed and achieved by re-purposing existing technology within the context of chipmusic is what draws me to this field, and has led me to undertake the current study. Discovering new uses for devices and laterally exploring technology via the creative means of musical expression has been an integral part of music technology and innovation. This remains true for chipmusic. It is through the development of technology that the non-musical is made creatively useful in a musical context, thus fashioning new and exciting opportunities for sonic expression, performance possibilities and music.

The aim of the current study is to give the community and interested individuals a process by which new chipmusic instruments can be developed, thereby offering composers and musicians a new palette of sound, specifically relating to chipmusic. This aim is achieved by enabling access to the sound chips inside certain obsolete video game systems that have either been ignored or under-utilised in the context of music making. Access to these sound chips is provided by a practical, methodological Interface Framework and has been applied to a number of video game consoles and discrete integrated circuits throughout the current study. The development and application of the Interface Framework has yielded practical results in terms of hardware and software, as well as creative results in terms of musical works. The methodology for the real time synthesis control of the musical output of a video game system is described, first in the generic sense (the structure of the Interface Framework) and then in detail for a number of specific cases (the application of the Interface Framework to a given video game system).

These processes have led to the creation of new musical instruments. Techniques from a range of areas, including microcontroller electronics, interface design, data mapping, assembly programming, synthesis and sample manipulation are combined to provide the music community with new technology for music-making as well as the *potential* for new technology. The practical outcomes of the current study provide composers and performers with new tools and new sounds that can be used in a creative context. The development and subsequent creative use of these tools is an exploration of a vein of technology, sonic abstraction and musical outcome that is a lateral investigation into machines from the past. This is done in the belief that these machines have not been explored to their full creative potential.

This submission is divided into three components - written, creative and data. The written component comprises a discussion detailing the key concepts and historical context of chipmusic; a presentation of the Interface Framework; four major case studies that document the application of the Interface Framework to four video game consoles and the results thereof; minor case studies and a conclusion. The four major case studies explore the development of Music Interfaces for the following video game consoles: SEGA Mega Drive, SEGA Master System, Atari 2600 and GCE / Milton Bradley Vectrex. The minor case studies section documents applications of the Interface Framework to the following integrated circuits: Phillips SAA-1099, Atari POKEY, Texas Instruments SN76489 and SP0256-AL2 A glossary of specialised terms is included in the written component at Appendix A.

The creative component features four works totalling approximately one hour of music. These works are an effective representation of the expression made possible via the technology developed throughout the current study. The creative component is presented in audio compact disc form, labelled Appendix D. All of the creative works have been composed, recorded and produced in the same fashion. Note data was entered into scenes and clips in Abletone Live, whereby the outputs of the relevant MIDI tracks were routed to the video game console sound chip. The audio output of the video game console was then routed back to the computer for monitoring during composition, and also for recording once the compositional process had taken place. Commonplace production techniques were employed on the resulting audio, including filtering and equalisation, multiband compression, limiting and level mixing. The creative works are *Antia* (a work for SEGA Mega Drive and Atari 2600), *Error Repeat* (a work for SEGA Master System and Nintendo Entertainment System), *Office Yoga Made Easy* (a work for SEGA Nomad and Atari 2600) and *Dynasty* (a work for the FM synthesis sound chip of the SEGA Master System). Appendix B provides written commentaries, graphical timelines and an audio compact disc track listing with track times relating to the creative works.

The data component disc (Appendix E) features all software-related facets of the current study. This includes microcontroller and microcomputer commented source code and compiled microcomputer applications (where appropriate) as well as Max/MSP patches. Documentation aimed at users is included for the developed Music Interfaces, as well as construction guides, electronic circuit schematics and sample packs featuring hardware recordings of the standalone integrated circuits. The creative works are included as uncompressed audio files in addition to timelines (either graphical or notational - as appropriate). The data component is presented on a data compact disc and is readable by Windows and Macintosh operating systems. Appendix C provides in hard copy the relevant schematics and printed circuit board designs – these also appear in the data component CD.

The method through which the results of the current study (both creative and practical) have been reached follows a specific form of research. An initial idea relating to musical control and interfacing in the field of chipmusic was first developed, tested and creatively trialled using a single video game console as a test case. Resources of time and money were allocated and a new skill set was learnt in order to bring this first example to fruition, from concept to a useable musical device. This method of control was then generalised and distilled to an Interface Framework whose processes and results can be duplicated when applied to more than one video game platform. It is in this way that multiple music interfaces can be developed from one overarching idea. Specific feedback via trial and error, experimentation and technical analysis was used to develop and tailor each interface to each video game console. However, the overall structure and form of each interface is practically identical.

The final proof of this research methodology lies in the creative application of the interfaces that have been developed. These musical works demonstrate and exemplify the level and type of control exercised by the use of the technology developed throughout the current study.

Overview of the Interface Framework

Conceptual Overview

The development of specialised, hardware-based Music Interfaces is undertaken by applying a generalised Interface Framework to obsolete video game consoles. In this context, the term ‘Music Interface’ refers to a device whose purpose it is to receive incoming music data, and send sound chip control data to a video game console. The Interface Framework is a combination of hardware and software, and is designed as a general approach to creating Music Interfaces for use in chipmusic. The Interface Framework allows for the external, musical control of the sound chip belonging to an obsolete video game console, such as the SN76489 sound chip within the SEGA Master System console. This control is achieved by the use of a custom programmed microcontroller that interprets general music data, such as note events, and subsequently outputs correlating audio control events, such as frequency or volume data, for a predetermined sound chip. The output of the microcontroller is connected to the video game console, which updates the internal sound chip accordingly. Thus, the video game console is able to output audio and music in response to incoming Musical Instrument Digital Interface (MIDI) data. The combination of hardware (i.e. microcontroller and related circuitry) and software (i.e. microcontroller and video game console applications) results in a Music Interface. The combination of such an interface with the video game console itself can be considered the creation of a new instrument. This Interface Framework is not limited to a particular obsolete video game console. Rather, it can be applied to a range of devices as demonstrated in the case studies explored through this research.

The Music Interfaces derived from the Interface Framework can be easily integrated into the modern computer music environment, as they use MIDI for communication and control of the video game system. The underlying principle is that a computer is used for composition, and music data is sent to the Music Interface in real-time via a MIDI connection. Alternatively, real-time control via physical controllers such as MIDI keyboards can be connected to the Music Interface (or to the computer) for the purpose of live performance.

The type of open-ended, external control advocated through this Interface Framework presents the end user with a number of advantages when contrasted with system-specific solutions for chipmusic composition. Real-time control of a video game console for music composition affords straightforward integration into modern music software packages, thus increasing the accessibility of chipmusic hardware. Furthermore, composers are not musically restricted by a graphical user interface that imposes specific structures for music making.

Consider the GUI (graphic user interface) and structural design of software such as Nanoloop for the Game Boy. Although Nanoloop is an excellent piece of software and has had some complex and beautiful music produced as a result, the restrictions of the structure of the music that can be created within the bounds of Nanoloop is clear when analysing a large set of music tracks that have been written using the software. This is a symptom of the role that the graphical user interface, software style and data limitations play within this context, rather than a symptom of the limitations set by the actual sonic capabilities of the Game Boy video game console.

For example, the composer is limited to using simple rhythmic durations to the exclusion of triplets and more exotic note durations. Each of the four channels within a given Nanoloop song is limited to a total of sixteen phrases. Every phrase has a maximum length of sixteen steps whereby each step is equal to a quaver, crotchet or minim. These two factors limit the complexity of musical structure, both on a note-by-note basis as well as the overall musical form. These limitations are not a product of hardware design or implementation but rather of software design and user interface.

Conversely, if a user has complete control over the sonic capabilities of a sound, the barrier of limitations of musical structure will be removed. By using an external source such as a host computer and appropriate MIDI music environment for the generation of control data, these limitations are overcome. For instance, algorithmic and generative music can be explored via the low-bit chip medium and an almost in-exhaustive amount of memory assures musical works of almost any length and complexity within the confines of the capabilities of the sound chip.

Although the Interface Framework has the advantage of unlimited compositional exploration as outlined above, there may be circumstances whereby existing software / hardware combinations may be more desireable. In the context of live performance, for example, a performer may be able to build a closer reporé with the audience by pressing the buttons of a Game Boy – and thus generating sound – rather than playing a music keyboard connected to a Game Boy via the Interface Framework.

Moreover, it might be argued that the limited resolution of MIDI data, which is in most cases 7 bits and usually not greater than 14 bits, might result in a control bottleneck when attempting to send data to a video game system. It should be noted that many computers and video game systems of the era in question are 8-bit, and many of their sound registers are also 8 bits in length. If there is a bottleneck, it should not be of a size that presents a problem. If any issues of controlling an 8 or 16 bit video game console using MIDI arise, they can be resolved through thoughtful mappings of musical events to sound chip data.

The Interface Framework is designed to be general enough so as to apply to a wide range of systems. Elements of the Interface Framework can be applied to discrete sound chips with some modification (See *Selected Discrete Sound Integrated Circuit Interface Designs* [pp. 79-83]). A number of factors are assumed when applying this Interface Framework to a given console. First, some method of storing and playing a program on the video game machine is required. This can be either an existing, original first-party cartridge with the ROM chip replaced using some form of re-programmable chip such as an erasable programmable read only memory (EPROM); or a third-party post market flash cart.

Flash carts are available for a range of systems including some video game consoles for which no music software has been written. However, a flash cart is often expensive in comparison to an EPROM-based approach. A possible benefit of using an EPROM is that the original ROM chip can be easily be de-soldered and replaced with an appropriate EPROM chip, although this technique may not be suitable in all cases. For instance, there may be too great a disparity between the pin configurations of the original video game cartridge memory chip and the EPROM chip that is to be used as a replacement.

The second factor that should be considered when applying the Interface Framework to a video game console is the availability of a software compiler for the target microcomputer system, and that enough documentation regarding the programming of the target microcomputer system is available. For example, the SEGA Master System features the Z80 microcomputer, which is well documented with a range of software compilers and debuggers available. This availability of documentation and tools relating to the Z80 microcomputer make the SEGA Master System the perfect candidate for the development of a Music Interface. To this end, the homebrew community (after market, user-based programmers) and fan clubs of obsolete computers and videogame hardware are able to supply technical information that is both relevant and highly detailed in many cases.

Finally, some basic soldering, maths and programming skills are desirable, although an absence of these skills will not prevent an intelligent and motivated individual from being able to apply the Interface Framework to a target video game or computer system.

Components of a Console-based Interface

Figure 1 summarises the flow of information with regard to receiving and mapping MIDI data to the video game console.

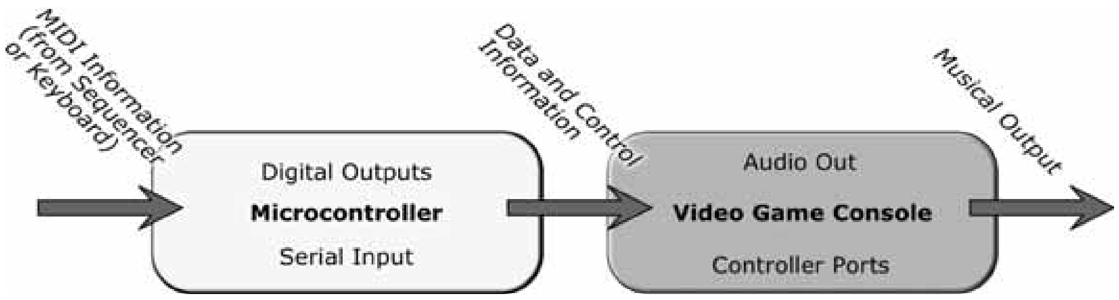


Figure 1: Overview of the Interface Framework

Microcontroller Background and Function

The microcontroller can be considered a translator or a conduit for data, receiving generic information (i.e. music data) and outputting relevant, specific information (i.e. data for a particular sound chip). Depending on the complexity of the sound chip, the microcontroller may be required to store the current state of the sound chip registers. Information is sent from a computer sequencer, MIDI keyboard or similar device and is received by a microcontroller running a custom program. The microcontroller is pre-programmed to interpret musical data, such as note pitch and note velocity, and to convert this information into a format that is recognisable by the video game console. Specifically, the way in which information is interpreted and represented by the microcontroller must in some way relate to the registers of the sound chip inside of the video game console. For example, the microcontroller might have a pre-programmed association of the pitch of a note with the appropriate frequency register inside the sound chip.

The extent to which the microcontroller has a memory of the current state of the sound registers of the video game console depends on the complexity of the sound chip being addressed. For example, the SEGA Mega Drive contains a complex FM synthesis chip with approximately 240 registers covering six channels, whereas its 8-bit predecessor – the SEGA Master System – has a four channel programmable sound generator (PSG) with only 8 registers[16]. In the case of the SEGA Mega Drive, it is necessary to keep a copy of the state of all sound registers, whereas this is not necessary with the SEGA Master System.

Certain characteristics were desirable when the microcontroller platform for the current study was selected. The microcontroller has been chosen based on the fact that it is inexpensive, easily programmed and powerful enough to handle an incoming data stream whilst communicating with the video game console in question.

Microcontroller Platform

The Arduino platform has been chosen as a microcontroller[17], as it features a suitable combination of a straight-forward programming language and cost-effective computation. These elements, combined with the near-ubiquity of the Arduino, make it the perfect candidate for interfacing. The term 'near-ubiquity' here refers to the amount and variety of different projects that have been developed using the Arduino in contrast with comparable

16 Yamaha Corporation. 'YM2612 Application Manual: Part I: Memory Map'. *http://www.smspower.org/maxim/Documents/YM2612#partimemorymap* (accessed 16 March 2010).

17 Arduino. *http://arduino.cc/* (accessed 16 March 2010).

microcontroller platforms such as the BASIC Stamp II and Picaxe.

The Arduino runs at 16MHz, and as such provides sufficient computational speed for any tasks required by a Music Interface. The amount of RAM and program memory available on the hardware is 2KB and 30KB respectively, which is also sufficient for all tasks in the case studies (excluding the storage of large, uncompressed audio files). Furthermore, the platform possesses sufficient digital outputs so as to communicate effectively with a video game. Additionally, the serial port has a variable baud rate and can reliably receive music data via a USB connection or a MIDI connection.[18]

The Arduino platform has a low part count for the most minimal, functioning circuit. This includes an ATmega328 microcontroller with the Arduino boot code, a 16MHz crystal, two 22pF capacitors and a 10k Ω resistor. Additionally, a 10uF electrolytic capacitor and a 0.1 uF ceramic capacitor may be required for smooth operation depending on the power source used. These capacitors are not required for battery-operated circuits.

This represents a very low base cost for electronic components and parts. Adding MIDI input functionality to this level of hardware increases the cost only marginally. Additional costs may include the manufacturing of a circuit board, casing and suitable video game controller port sockets or plugs and related wiring. However, all of these additional costs are independent of the chosen microcontroller or computational platform.

The Video Game Console

The microcontroller transfers data to the video game console, which uses what is often the only means of communicating with the outside world to receive that information: the controller ports. These ports are used for game play under normal circumstances, but can also be used to transmit and receive data.

18 'Arduino: Arduino Board Duemilanove'. http://arduino.cc/en/Main/ArduinoBoardDuemilanove (accessed 16 March 2010).

The video game console is ‘aware’ of the type of data and format to expect on its controller ports from the microcontroller, as the custom software console is continuously polling for data and control information. Because the data and control format is identical to the way in which the microcontroller sends its information, data integrity can be assured. Once the video game console’s microcomputer has successfully received this data, the information is sent on to its sound chip. The sound chip then changes its output due to the data, thus resulting in audio that has been generated in response to MIDI data received by the microcontroller.

Data Mapping

A number of musical parameters may require more than one transmission of data from the microcontroller to the video game console. For instance, if the microcontroller receives music data that indicates a change in pitch, two or more write cycles may be required to update the frequency of the sound chip inside of the video game console. More than one 8-bit register may be combined in order to control the frequency of a given sound generator inside of a sound chip.

The number of unique frequencies that a video game console can reproduce depends on the specifications of the sound chip, with typical values ranging from 1,024 possible pitches for the SEGA Master System[19] to 65,536 possible pitches for the Atari POKEY chip. However, there are exceptions. For example, the Atari 2600 can only reproduce 32 different pitches for each of its distortion settings, making it a very limiting and challenging device to write music with.

In general, the mapping of frequency to pitch is straightforward, as a given MIDI

19 Texas Instruments Semiconductor Group. SN76489 Datasheet. http://soundprogramming.net/electronics_reference/manuals/SN76489datasheet.pdf (accessed15 March 2010).

note should correspond to a given frequency as closely as is technically possible. However, there are more abstract considerations to be made when mapping other musical data to the control of a sound chip. For instance, how should timbre, noise type or filter be mapped to MIDI data? These are aspects that need to be considered with regard to potential users, the intended musical use and the audio and computational resources available to the microcontroller and the video game console.

Typically, the information that is sent from the microcontroller to the video game console is a target register to select which sound parameter to change, and the data value that is stored in the selected target register. Once the system has received both the register and the value, the value can be written by the video game console's microcomputer to the sound register inside of the sound chip. This generalisation is not applicable to all sound chips. For example, the SN76489 only has a single 8-bit data bus without any traditional address selection. In this case, parameters are selected by means of data latching and specific bit combinations on the data bus.

The Process of Applying the Interface Framework

The core concept of the Interface Framework is that the sound chip of an obsolete video game console can be controlled in real-time via a combination of two pieces of software that interact with one another through a piece of hardware. The hardware in question is the Arduino microcontroller and related electronic components. Residing onboard the microcontroller is one of the two pieces of software. This software / hardware combination is responsible for interpreting MIDI music data and responding to that data by sending control information via the controller port(s) of the video game console. The second software element, residing in the memory of the video game console, is responsible for accepting incoming control information and then routing that data to a sound chip. This core idea of two software components and one hardware component can be applied to a variety of video game machines. The process of applying this Interface Framework to a given console can be generalised in a number of steps that result in the creation of a musical interface.

The development process is the reverse of the flow of information. That is, first the software for the video game console should be developed, followed by the hardware for the interface and finally the software for the microcontroller. The simplest components of the chain (i.e. the act of moving data from the controller ports to the sound circuits) are developed first, as the more complex elements (i.e. the mapping of general music data to specific sound chip control data) rely on the former in order to function correctly and be tested.

The first stage of the development process requires knowledge and tools relating to the video game console in question. A compiler for the target microcomputer is needed, as well as the relevant memory maps, input / output port descriptions and microcomputer register designation as well as an instruction set. Additionally, documentation regarding the sound chip of the video game console is required, as well as some method via which a custom compiled program can be loaded into the video game console.

A protocol is devised whereby the correct amount of data corresponding to the sound chip's number of registers and register lengths can be transferred via the controller ports. The aim of the protocol is to provide a simple pathway from the controller ports to the sound chip, allowing for the transparent control of the sound chip. The protocol will include a write signifier bit (being an indicator of data present on the controller ports) as well as data bytes and, if required, nibble and address signifier bits. However, the exact nature of the protocol implemented is dependent on the architecture of the sound chip as well the amount of information that can be sent via the controller ports. Each interface requires a custom protocol to allow data to be transferred from the microcontroller to the video game console. The specifications of a console and a sound chip will result in a different protocol for each interface, although a few overall concepts are still applied. The way in which data is sent from the microcontroller to the video game console is often in smaller chunks such as nibbles (four bits). Consider the controller port data from an imaginary video game system as it is used during game play, summarised in Table 1.

Bit	Button
7	Up
6	Down
5	Left
4	Right
3	Action Button 1
2	Action Button 2
1	Action Button 3
0	Action Button 4

Table 1: Hypothetical controller port data

Assuming that the sound chip of the target video game console has 16 registers and each register has a width of 8 bits, it is possible to define a protocol with which data can be quickly and efficiently sent to the system, an example of which is given in Table 2.

Bit	Protocol Function
7	Write Signifier
6	Nibble Type
5	Address
4	- (unused)
3	Nibble bit 3
2	Nibble bit 2
1	Nibble bit 1
0	Nibble bit 0

Table 2: Hypothetical write protocol

Bits 0 to 3 represent the actual data that is being transferred. Bit 4 is not used. Bit 5 determines whether the nibble currently being received is part of the data value or the address value. If this is low, then the nibble belongs to the data value. If this is high, then the nibble corresponds to the address. Bit 6 is the nibble type and is only applicable for data bits (as only a single nibble is required for indicating the address). If this is low, then the nibble received is a low data nibble. If this is high, then the nibble received is a high data nibble. Bit 7 is the write signifier. If this is high, then the console is informed that legitimate data is present on the remaining bits on the controller port.

One complete write cycle would consist of the microcontroller writing data with the appropriate address bit and nibble type bit set. In total, 3 nibbles need to be written using 3 separate data transfers – one nibble for the lower half of the data value, one for the upper half of the data value and one nibble for the address value. The target microcomputer will need to store the values of the nibbles as each cycle progresses. Between each transfer of data, an idle state must be defined whereby no data is written / read. Once the address nibble has been received, the target microcomputer can transfer the combined high and low nibbles of the data value as a byte to the sound chip register specified by the address nibble. Each of these four states (whether no data / address is being written; whether the low nibble of data is being written; whether the high nibble of data is being written; and whether the address nibble is being written) is summarised in Table 3, Table 4, Table 5 and Table 6 respectively.

Bit Number	Protocol Function	Bit Value	Bit Value description
7	Write Signifier	LOW	No valid data is present – therefore, no write / read will take place
6	Nibble Type	HIGH / LOW	Of no consequence – no write / read will take place
5	Address	HIGH / LOW	Of no consequence – no write / read will take place
4	- (unused)	-	-
3	Nibble bit 3	HIGH / LOW	Of no consequence – no write / read will take place
2	Nibble bit 2	HIGH / LOW	Of no consequence – no write / read will take place
1	Nibble bit 1	HIGH / LOW	Of no consequence – no write / read will take place
0	Nibble bit 0	HIGH / LOW	Of no consequence – no write / read will take place

Table 3: Hypothetical protocol - no data is written.

Bit Number	Protocol Function	Bit Value	Bit Value description
7	Write Signifier	HIGH	Valid data is present
6	Nibble Type	LOW	Indicates that a low data nibble is present
5	Address	LOW	The incoming nibble is part of a data byte
4	- (unused)	-	-
3	Nibble bit 3	HIGH / LOW	Data value bit 3
2	Nibble bit 2	HIGH / LOW	Data value bit 2
1	Nibble bit 1	HIGH / LOW	Data value bit 1
0	Nibble bit 0	HIGH / LOW	Data value bit 0

Table 4: Hypothetical protocol - writing the low data nibble

Bit Number	Protocol Function	Bit Value	Bit Value description
7	Write Signifier	HIGH	Valid data is present
6	Nibble Type	HIGH	Indicates that a high data nibble is present
5	Address	LOW	The incoming nibble is part of a data byte
4	- (unused)	-	-
3	Nibble bit 3	HIGH / LOW	Data value bit 7
2	Nibble bit 2	HIGH / LOW	Data value bit 6
1	Nibble bit 1	HIGH / LOW	Data value bit 5
0	Nibble bit 0	HIGH / LOW	Data value bit 4

Table 5: Hypothetical protocol - writing the high data nibble

Bit Number	Protocol Function	Bit Value	Bit Value description
7	Write Signifier	HIGH	Valid data is present
6	Nibble Type	HIGH or LOW	Of no consequence, as the address is only a single nibble
5	Address	HIGH	The incoming nibble is an address
4	- (unused)	-	-
3	Nibble bit 3	HIGH / LOW	Address value bit 3
2	Nibble bit 2	HIGH / LOW	Address value bit 2
1	Nibble bit 1	HIGH / LOW	Address value bit 1
0	Nibble bit 0	HIGH / LOW	Address value bit 0

Table 6: Hypothetical protocol - writing the address nibble

Once such a protocol has been devised, appropriate software can be written for the video game console that accepts data from the controller ports and sends it to the sound chip. This involves creating a loop of code whereby the controller ports are continuously checked for the relevant write signifier bit. If the test for the write signifier bit is positive, then data from the joystick ports can be read, and, depending on the complexity of the protocol, either stored for further use or sent directly to the sound chip. Note that in many cases, the process of receiving data via such a protocol will involve more than one cycle of reading the state of the joystick ports. In these cases, data may be stored in RAM or in a computer register before being reconstructed into information that is intended for the sound chip. Following the development and compiling of the software for the video game, a flash cart or modified cart can be used to run the program on the hardware.

A hypothetical example of assembly language code for a hypothetical video game console relating to the data in Table 1 and the subsequent protocol from Table 3 is provided below. Note that each commented line is preceded with a semicolon.

```
main:
        ; the main program starts here
        PORT                EQU    $7F
        ; create a definition for the joystick port location
        SOUND_CHIP_ADD      EQU    $80
        ; create a definition for the sound chip address location
        SOUND_CHIP_DAT      EQU    $81
        ; create a definition for the sound chip data location
        ; main loop begins here - checking for data at joystick ports
```

```
loop:
        ; the main loop starts here
        LDA     PORT
        ; load the data from the joystick port into register A
        BITA    %10000000
        ; check bit seven of the joystick port (the write signifier bit)
        BEQ     loop
        ; is the bit high? if so, keep going. if not, branch back to the loop

write_cycle:
        ; the write cycle begins here
        BITA    %00100000
        ; check bit five of the joystick port (the address bit)
        BEQ     data_type
        ; is the bit high? if not, the data received is a data nibble
        ; write cycle three begins here - address write cycle
address_type:
        ; if yes, the data received is an address
        ANDA    $0F
        ; discard the upper four bits of the joystick data
        STA     SOUND_CHIP_ADD
        ; send the address (from register A) to the sound chip
        STB     SOUND_CHIP_DAT
        ; send the data byte (from register B) to the sound chip
        BRA     loop
        ; branch back to the main loop and wait for the next set of write cycles
data_type:
        ; a data type has been received
        BITA    %01000000
        ; check bit six of the joystick port (the nibble bit)
        BEQ     low_nibble
        ; if the bit is low, the nibble received should be low

        ; write cycle two begins here - high nibble data write cycle

high_nibble:
        ; if the bit is high, the nibble received should be high
        ANDA    $0F
        ; discard the upper four bits of the joystick data - only want the nibble
        LSL
        ; shift the nibble to the left by one bit
        LSL
        ; shift the nibble to the left by one bit
        LSL
        ; shift the nibble to the left by one bit
        LSL
        ; shift the nibble to the left by one bit
        ORB
        ; combine the low nibble (received previously) with the high nibble
        TFR
        ; transfer register A to B - now, the entire data byte is in register B
        BEQ     loop
        ; branch back to the loop and wait for the address write cycle
        ; write cycle one begins here - low nibble data write cycle

low_nibble:
        ANDA    $0F
        ; discard the upper four bits of the joystick data - only want the nibble
        LDB     #0
        ; clear register B
        TFR
        ; transfer register A to B - now, the low data nibble is in register B
        BEQ     loop
```

; branch back to the loop and wait for the high nibble write cycle

The method by which data is passed to the sound chip can be tested by connecting the microcontroller to the video game console via the controller port(s). A simple test program can be written for the microcontroller that transmits data to all registers of the sound chip via the protocol as defined earlier. If the audio output of the video game console resembles what is to be expected based on the register, then the process of writing data to the sound chip is successful. However, if the audio output of the video game console is unresponsive, it is necessary to re-examine the protocol and its implementation on both the micrcontroller and the video game console. Below is a hypothetical example of Arduino microcontroller source code that transmits data via the protocol as listed in Table 3. Note that each commented line is preceded with two forward slashes.

```
// TEST PROGRAM (RANDOM DATA)
// create a variable to store sound chip address data
byte address;
// create a variable to store sound chip register data
byte data;
// define the write pin
int write_pin = 2;
// define the nibble pin
int nibble_pin = 3;
// define the address pin
int address_pin = 4;

// set up the microcontroller
void setup() {
        // set data port to output
        DDRC = DDRC | B0011111111;
        // set pin to output
        pinMode(write_pin, OUTPUT);
        // set pin to output
        pinMode(nibble_pin, OUTPUT);
        // set pin to output
        pinMode(address_pin, OUTPUT);
        // create a random seed for random number generator
        randomSeed(analogRead(5));
}

void loop() {
        // generate a random number between 0 and 15 – this will be the address
        address = random(16);
        // generate a random number between 0 and 255 – this will be the register data
        data = random(256);
        // call the routine that actually writes data to the video game console – see below for
        the "writeData" routine
        writeData(address, data);
        // wait for a moment
        delay(20);
        // loop forever
}

void writeData(byte address, byte data) {
        // write cycle one begins here - low data nibble
        // first, set an output port to the low nibble of the data byte
        PORTC = data & B00001111;
        // set the address pin low, as this is a data nibble
        digitalWrite(address_pin, LOW);
        // set the nibble pin low, as this is a low data nibble
        digitalWrite(nibble_pin, LOW);
        // set the write pin high, as the data is now ready
        digitalWrite(write_pin, HIGH);
        // wait for a short moment so that the target microcomputer can load the data
        delayMicroseconds(10);
        // set the write pin low, as the data has been transferred
        digitalWrite(write_pin, LOW);
        // wait for another moment so that the target microcomputer can deal with the data
        delayMicroseconds(10);
        // write cycle two begins here - high data nibble
        // first, set an output port to the low nibble of the data byte
        PORTC = data << 4;
        // set the address pin low, as this is a data nibble
        digitalWrite(address_pin, LOW);
        // set the nibble pin high, as this is a high data nibble
        digitalWrite(nibble_pin, high);
        // set the write pin high, as the data is now ready
        digitalWrite(write_pin, HIGH);
        // wait for a short moment so that the target microcomputer can load the data
        delayMicroseconds(10);
        // set the write pin low, as the data has been transferred
        digitalWrite(write_pin, LOW);
        // wait for another moment so that the target microcomputer can deal with the data
        delayMicroseconds(10);

        // write cycle three begins here - address data nibble
        // first, set an output port to the low nibble of the data byte
        PORTC = address;
        // set the address pin high, as this is the address nibble
        digitalWrite(address_pin, HIGH);
        // set the write pin high, as the address is now ready
        digitalWrite(write_pin, HIGH);
        // wait for a short moment so that the target microcomputer can load the address
        delayMicroseconds(10);
        // set the write pin low, as the data has been transferred
        digitalWrite(write_pin, LOW);
        // wait for another moment so that the target microcomputer can deal with the address
        delayMicroseconds(10);
        // wait for another moment so that the target microcomputer send the data and address
        to the sound chip
        delayMicroseconds(10);
}
```

Once successful transmission of control data from the microcontroller to the console has taken place, the protocol and the method of transmission can be optimised. An inefficient method may place unwarranted overheads on the amount of time required to transmit and subsequently receive data, which in turn will negatively affect the expressiveness of the interface. Efficiency of transmission can be increased by practically examining the most suitable timing for each step of the interface. This process of transmitting data can then be encapsulated as a function that can be called during the main loop of the software. In the above hypothetical example of Arduino microcontroller source code, this encapsulation can be seen as the function *void writeData(byte address, byte data) {}*.

From this point, the mapping of musical events to sound chip parameters can be developed. This is achieved by writing a set of routines for the microcontroller that make connections between specific MIDI messages, such as pitch and velocity, with the appropriate sound chip parameters, such as frequency and volume, using the encapsulated writing functions. The complexity of such routines and how they function is largely dependent on the nature of the sound chip to which data is to be written.

MIDI continuous controllers (MIDI CC) are channel-specific MIDI messages that should be associated with global and channel-specific parameters of the sound chip such as filter, timbre and panning controls. The extent to which a given sound chip will require MIDI CC mappings varies greatly from model to model. This process of mapping MIDI CC should be repeated until all aspects of the sound chip are under the control of the interface.

The mapping should be tested and re-designed as necessary. Once the sound chip is sufficiently and accurately controlled, the process of extending control over that sound chip can be exploited. This includes possibilities such as the modulation of volume data to achieve sample playback and software-based synthesis, as well as timbral modulation. The extent to which such features can be implemented depends on both the characteristics of the sound chip and the speed of the protocol (i.e. sufficient speed of data transmission is required for sample playback).

The Interface Framework has been applied to a number of consoles and sound chips, and as such is a proven framework with working results. Consoles that have had Music Interfaces developed within the context of the Interface Framework include the GCE / Milton Bradley Vectrex, the SEGA Master System, the SEGA Mega Drive and the Atari 2600. Sound chips that have had Music Interfaces built for them include the General Instruments SP0256-AL2, the Texas Instruments SN76489, the Atari POKEY and the Phillips SAA1099.

These consoles and chips were designed by a number of manufacturers from 1978 to 1989. A number of the sound chips and consoles listed have not been used directly in a chipmusic context. By applying this Interface Framework, a new palette of sounds and new ways of creating chipmusic will be achieved for the wider music technology community. Importantly, the Interface Framework as previously described can be applied to a wider range of consoles and chips than is present in the case studies.

What follows are a series of case studies that document the application of the Interface Framework to four consoles – resulting in Music Interfaces – as well as four standalone integrated sound chips. In the major case studies, a brief background is given for each video game console as well as the sonic properties inherent to the sound chips and the detailed application of the Interface Framework. Furthermore, advanced techniques designed to extend the sonic output and increase the variety of musical expression of each console via each Music Interface are discussed.

SEGA Mega Drive / Genesis Music Interface

SEGA Mega Drive / Genesis Console Overview and Background

Known as the SEGA Genesis in North America, the 16-bit SEGA Mega Drive video game console was released in 1987 and enjoyed relative commercial success into the mid-1990s.[20] The console distinguished itself from competitors by featuring fast-paced action game play, a greater range of sports-oriented games and add-ons, including a compact disc-drive peripheral.

Interestingly, the SEGA Mega Drive / Genesis continues to be popular in the developing world, as second-hand obsolete consoles are easier to come by and are more affordable than their current-generation counterparts. As a result of a resurgence in popularity of SEGA Mega Drive / Genesis titles on current-generation video game consoles, such as the Microsoft XBox 360 and the Nintendo Wii, a number of licensed and unlicensed SEGA Mega Drive / Genesis clones have appeared on the market in recent years. All of these clones work on the basis of single-chip emulation, whereby the functions of the many integrated circuits found within the original console are handled by only a small number of components. Sound quality suffers as a direct consequence of this form of emulation, not to mention artefacts and strange behaviours such as incorrect pitch, monophonic playback and sample rate issues.

Sound Capabilities of the Console

The SEGA Mega Drive console contains two chips for sound production; the primary one being the Yamaha YM2612 FM (frequency modulation) chip, which is explored here. The secondary chip is the SN76489, which is integrated as part of the visual display processor.

20 Mark Wolf. *The Video Game Explosion: A History from PONG to Playstation and Beyond* (Santa Barbara: Greenwood Publishing, 2007), 149.

The latter, which was included with the SEGA Master System console in order to facilitate backwards compatibility with older 8-bit consoles, is discussed in the case that follows. [21]

The YM2612 is a six-channel FM synthesis generator. It was the first Yamaha FM synthesis chip to feature an onboard analogue to digital converter, which reduced the complexity of the circuit in comparison to its predecessors. Each channel of the YM2612 has a set of four sine wave operators. The configuration of the operators – which dictates which is acting as a modulator and which is acting as a carrier – is set by one of eight preset algorithms, and can be independently controlled on a channel-by-channel basis. The eight algorithms are shown below in Figure 2.

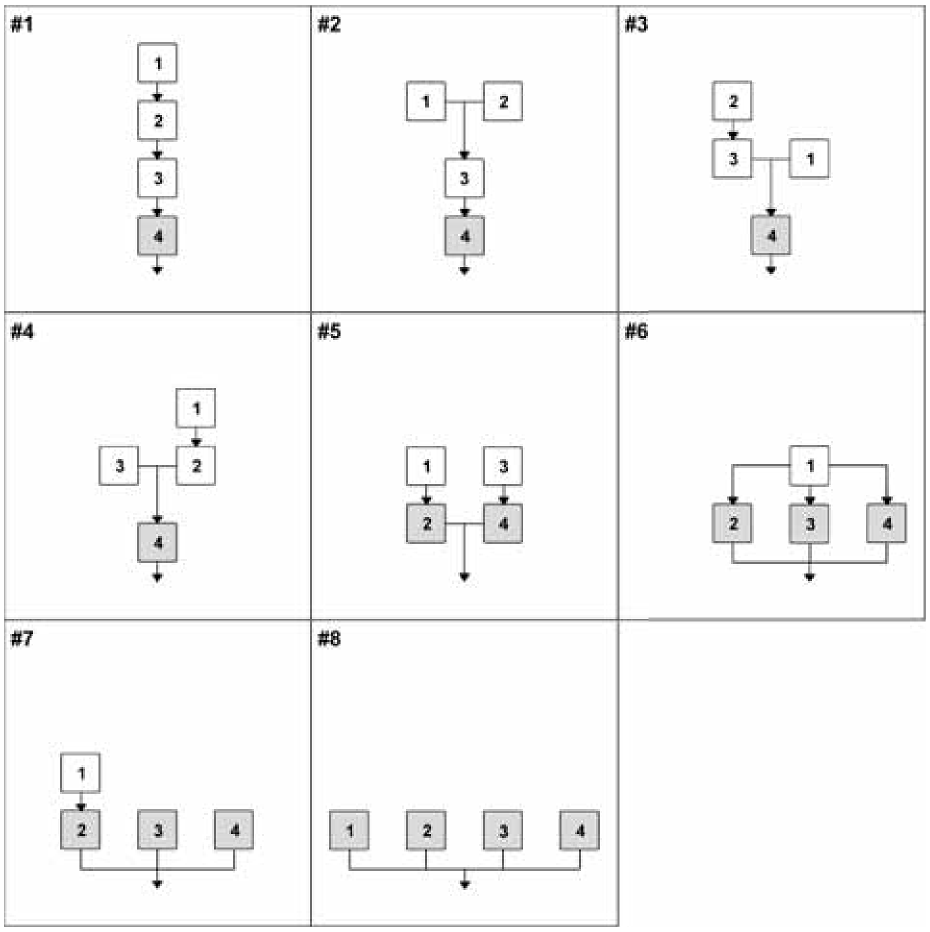


Figure 2: YM2612 FM operator algorithms

21 Yamaha Corporation. YM2612 Application Manual: Overview, at http://www.smspower.org/maxim/Documents/YM2612#overview (accessed 23 March 2010).

In the diagram above a grey box indicates that a given operator is acting as a carrier and is therefore directly audible. A white box indicates that a given operator is acting as a modulator and is controlling the frequency of another operator.

Each operator has an amplitude envelope consisting of attack, decay one, amplitude one, decay two, amplitude two and release sections. This complex envelope shape gives rise to a range of synthesised sounds. Additional parameters can be controlled, including the frequency multiple and detune amount for each operator, as well as the feedback amount between operator 4's output and input control signal. YM2612 tuning is precise, as a 14-bit register governs frequency.

Although all channels are identical in their basic functions, channels 3 and 6 have additional features. The frequency of the individual operators that comprise channel 3 can be set independently; this is not the case with the other channels, where the frequency relationship between operators and the channel's fundamental frequency is limited to integer multiples. Channel 6 can act as a digital to analogue converter for 8-bit digital samples, which gives the SEGA Mega Drive the ability to play back digitised sounds. Finally, there are a number of global low-frequency oscillator parameters that can modulate the frequency and amplitude of a given channel, which can be panned hard left, hard right or centre.

A less complex chip than the YM2612, the SN76489 comprises three square wave channels with limited pitch range and frequency resolution, as well as one noise channel. Although the music and synthesis characteristics of the YM2612 and SN76489 are theoretically identical, regardless of the model of SEGA Mega Drive employed, there is a significant amount of variation in the sonic output of various SEGA Mega Drive / Genesis models, including SEGA Mega Drive / Genesis 1, SEGA Mega Drive / Genesis 2, the SEGA Nomad and various SEGA Mega Drive / Genesis clones. The sonic differences between these models include output volume levels, signal to noise ratios, the bands of noise present in an empty signal, distortion levels, bass and treble response, and the level of distortion in the SN76489's output.

Current Capacity for Music-making on the Console

There are a number of methods that can be used to compose music for the SEGA Mega Drive system, yet all of those currently available have drawbacks. A widely used method is the TFM Music Maker[22], a tracker and sound editor for the Windows operating system that is able to output music files compatible with the OPN family of Yamaha FM synthesis sound chips, including the SEGA Mega Drive YM2612.

Although TFM Music Maker is the standard tool for a number of chipmusic composers who use the SEGA Mega Drive / Genesis sound, it too has limitations. The first of these is that the tracker itself is not native to the SEGA console, but rather runs in a host operating system (in this case Windows). As a result, the tracker outputs data files. Second, there is a certain amount of complexity involved in playing the sounds back on actual SEGA Mega Drive / Genesis hardware. For instance, the playback of music would require a custom, rewriteable SEGA Mega Drive / Genesis cartridge, which is relatively rare (in comparison to Game Boy cartridges) and only manufactured in small batches by three sources. One of the three types of cartridges available requires the use of the obsolete parallel port, whereas the other two support various methods of more modern USB-based transfer. The software used to write data to these cartridges is only available for the Windows operating system.

Once a TFM Music Maker music file has been written to the cartridge it cannot be changed unless a new version is used to overwrite the obsolete music file. It follows that

22 Alex Semenov. TFM Music Maker (software for Windows OS, 2009) ftp://ftp.untergrund.net/users/shiru/tfmmaker14.zip (accessed 23 March 2010).

no generative procedures, real-time synthesis control or real-time performance can be utilised in conjunction with the SEGA Mega Drive when making music using the TFM Music Maker software package. The third limitation is that the TFM Music Maker, being centred on the OPN family of Yamaha FM synthesis sound chips, only supports the YM2612 and not the SN76489 PSG sound chip. This calls into question the authenticity and completeness of music made using the TFM Music Maker. In addition to TFM Music Maker, the SEGA G.E.M.S. environment also provides an opportunity for music composition on the console, albeit with many of the same shortcomings such as linear structure and no real-time control over the outcome.

Music can also be composed for the SEGA Mega Drive / Genesis is by using various types of FM synthesis emulators and audio software plugins. However, it must be noted that although this method can come close to emulating sonic archetypes employed in some Mega Drive / Genesis games, there is currently no plugin that is an emulator of the YM2612. As a result, it is categorically impossible to fully emulate the sounds of the SEGA Mega Drive / Genesis via the method of audio host-based plugins, although the Plogue Chipsounds audio plugin includes an excellent virtualisation of the SN76489 chip. In any case, emulation-based plugins will be incapable of reproducing the nuances of music played back on real-world hardware.

Directly programming a SEGA ROM file in either 68000 assembly or in the C programming language is also possible. But in doing any attempt at writing for the SEGA Mega Drive / Genesis will suffer from generative shortcomings similar to those of the TFM Music Maker. The section that follows here describes the development of a music interface that overcomes these and other shortcomings.

SEGA Mega Drive / Genesis Music Interface: Concept, Design and Capabilities

The SEGA Mega Drive / Genesis Music Interface has been created as a result of applying the Interface Framework to the SEGA console. In the case of the SEGA Mega Drive / Genesis Music Interface, the player 2 controller port of the SEGA Mega Drive / Genesis is used for controlling the audio synthesis processes. The YM2612 has more than 240 registers spread over 2 register ‘pages’. One page is responsible for voices one to three, while the second page is responsible for voices four to six.

Data is transferred from the microcontroller to the SEGA Mega Drive on a nibble-by-nibble basis and, as such, a total of four write events are required before a register is updated. These four write events represent:

- The four low data bits of the address byte.
- The four high data bits of the address byte.
- The four low data bits of the data byte.
- The four high data bits of the data byte.

The process of transferring data is handled by the 7 data pins that make up the second controller port of the console. Four data bits represent the actual nibble of data to be transferred, irrespective of whether that nibble is part of a data byte or an address byte. The remaining 3 pins signify the data type (i.e. whether address or data), the nibble type (i.e. whether high or low) and whether accurate data is present (essentially a 'WRITE' pin). This process is summarised in Table 7, below.

Controller Port Pin	**Description**	**Protocol Function**	**68K Location**
1	Up	DATA BIT 0	$A10005, bit 0
2	Down	DATA BIT 1	$A10005, bit 1
3	Left	DATA BIT 2	$A10005, bit 2
4	Right	DATA BIT 3	$A10005, bit 3
5	5V	N/A	N/A
6	Input 1	NIBBLE	$A10005, bit 4
7	Output	WRITE	$A10005, bit 6
8	0V	N/A	N/A
9	Input 2	TYPE	$A10005, bit 5

Table 7: SEGA Mega Drive / Genesis controller port pins, functions and locations

Theoretically, given that the address length of 8 bits translates into 256 values, the above SEGA Mega Drive / Genesis Music Interface protocol allows the interface to interact with up to 256 separate addresses. However, as each page within the sound chip contains less than 128 addresses there is spare ‘space’. This space is used to determine which register page should be written to, and also allows the SEGA Mega Drive / Genesis Music Interface to write data to the SN76489 programmable sound chip. The latter process is achieved by setting variables which the 68K stores for the next round of data. For example, if data is to be written to the SN76489 sound chip, an address value of 192 is sent to the 68K. The software executed by the SEGA Mega Drive / Genesis filters out this particular address number, as the YM2612 only contains 120 registers per page. As a result, the data value accompanying the ‘fake’ address byte is written to the SN76489 as opposed to the YM2612.

In a similar fashion, an address value of 193 sets a flag for the first page of YM2612 registers, whilst a value of 194 sets a flag for the second page. These flags are latching values, and as such only need to be written to when a *change* in page is required, as opposed to every single register write. For more details refer to the microcontroller and microcomputer source code on the data disc (Appendix E), located at Major Case Studies/SEGA Mega Drive/Source Code/.

From a hardware perspective, the SEGA Mega Drive / Genesis Music Interface consists of a microcontroller (ATmega328), an optocoupler, one 5 pin DIN MIDI connection, one 9 pin D-subminiature connection , and a small number of passive components such as resistors, capacitors and a diode. The schematic can be viewed on the data disc (Appendix E) in the location Major Case Studies/SEGA Mega Drive/Schematic. Figure 3 shows an example of a printed circuit board design that encompasses the hardware portion of the SEGA Mega Drive / Genesis Music Interface.

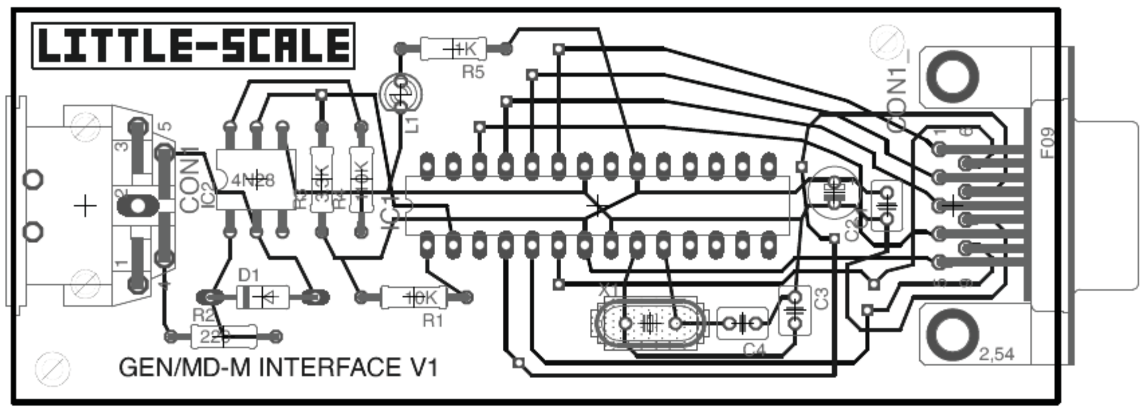


Figure 3: Single 9 pin D-subminiature MIDI Interface Hardware PCB

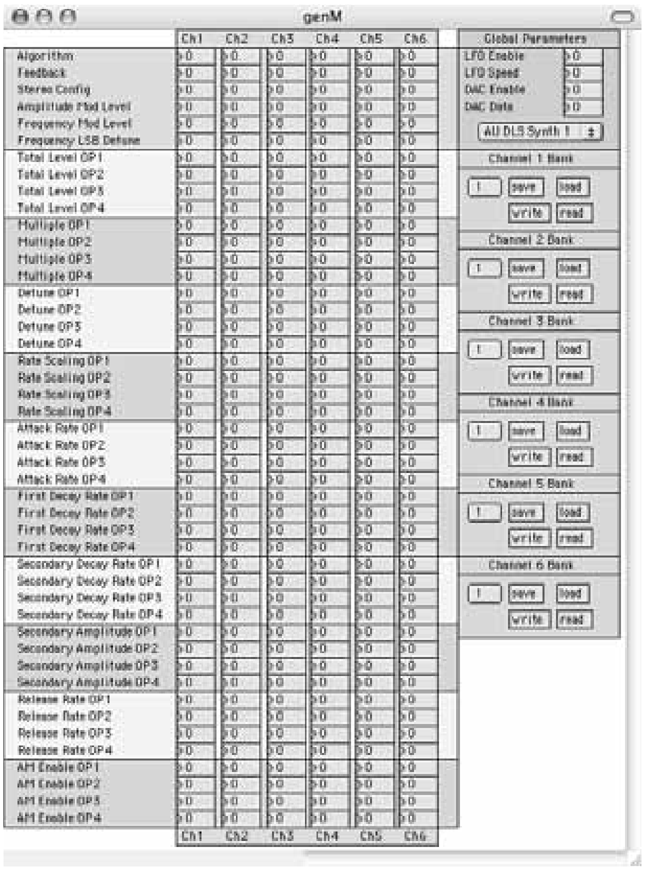
One issue faced during the software design of the SEGA Mega Drive / Genesis Music Interface was how to develop appropriate frequency and tuning mappings. A 14-bit frequency register governs the frequency of an FM channel, and is divided into two parts – the block and the data. The block portion of the frequency register is 3 bits in length and controls the octave of a given channel's frequency. The data portion is 11 bits in length and controls the actual frequency within the range of a given block. The frequency range within a given block is larger than one octave and, as a result, the block portion of the data should not simply be considered the most significant 3 bits of the frequency as a whole.

The challenge was that the relationship between data and tuning is not documented in the YM2612 technical documentation, which simply lists a set of data values from which to derive the tuning of an equally tempered scale. For the purposes of a musical interface a list of values is, in any case, insufficient for constructing a worthwhile mapping of data to musical sound. By researching a mathematical relationship between data value and frequency, a more complex mapping more suited to the characteristics of the chip is possible. The tuning of the SEGA Mega Drive / Genesis Music Interface proposed here was developed by compiling a list of known frequencies and YM2612 data values. The list, correct for every data block, was analysed, and from this an exponential function describing the musical relationship between frequency and data was created. The exponential function was in turn used to map MIDI notes to frequency data.

The benefits of deriving frequency from a mathematical function as opposed to a standard look-up table are two-fold. First, it easily accommodates tunings based on octave divisions over and above the number 12. As a result, the interface is capable of switching between any equal tempered tuning from divisions of 4 to 40, a unique feature when compared to the vast majority of chipmusic hardware devices and related software. The second benefit centres on the use of a function for pitch derivation, which assists in creating a more musical mapping between MIDI pitch-bend data and changes in pitch. Conversely, if the method of data value determination for frequency is via a lookup table, changes in data values of less than a semitone can result in an unmusical frequency output. For example, a pitch bend value of + 200 will make a pitch sharper when applied to a note with a pitch identity of B when compared to a pitch identity of G, because of the way that frequency is represented and the correlation between pitch and data within the YM2612.

Due to the complexity of the YM2612 chip, an FM patch editor named ‘genM’ has been developed (see Figure 4) alongside the SEGA Mega Drive / Genesis Music Interface in order to allow users an easy way to explore the sonic possibilities of the FM synthesis.

The patch editor, source code and documentation can be found on the data disc (Appendix E) in the location Tools/genM. Information regarding the exact nature of the mapping of MIDI data to the YM2612 sound chip can be found in the SEGA Mega Drive / Genesis Music Interface, which is on the data disc (Appendix E) in the location Major Case Studies/SEGA Mega Drive/User Guide.



	Ch1	Ch2	Ch3	Ch4	Ch5	Ch6
Algorithm	0	0	0	0	0	0
Feedback	0	0	0	0	0	0
Stereo Config	0	0	0	0	0	0
Amplitude Mod Level	0	0	0	0	0	0
Frequency Mod Level	0	0	0	0	0	0
Frequency LSB Detune	0	0	0	0	0	0
Total Level OP1	0	0	0	0	0	0
Total Level OP2	0	0	0	0	0	0
Total Level OP3	0	0	0	0	0	0
Total Level OP4	0	0	0	0	0	0
Multiple OP1	0	0	0	0	0	0
Multiple OP2	0	0	0	0	0	0
Multiple OP3	0	0	0	0	0	0
Multiple OP4	0	0	0	0	0	0
Detune OP1	0	0	0	0	0	0
Detune OP2	0	0	0	0	0	0
Detune OP3	0	0	0	0	0	0
Detune OP4	0	0	0	0	0	0
Rate Scaling OP1	0	0	0	0	0	0
Rate Scaling OP2	0	0	0	0	0	0
Rate Scaling OP3	0	0	0	0	0	0
Rate Scaling OP4	0	0	0	0	0	0
Attack Rate OP1	0	0	0	0	0	0
Attack Rate OP2	0	0	0	0	0	0
Attack Rate OP3	0	0	0	0	0	0
Attack Rate OP4	0	0	0	0	0	0
First Decay Rate OP1	0	0	0	0	0	0
First Decay Rate OP2	0	0	0	0	0	0
First Decay Rate OP3	0	0	0	0	0	0
First Decay Rate OP4	0	0	0	0	0	0
Secondary Decay Rate OP1	0	0	0	0	0	0
Secondary Decay Rate OP2	0	0	0	0	0	0
Secondary Decay Rate OP3	0	0	0	0	0	0
Secondary Decay Rate OP4	0	0	0	0	0	0
Secondary Amplitude OP1	0	0	0	0	0	0
Secondary Amplitude OP2	0	0	0	0	0	0
Secondary Amplitude OP3	0	0	0	0	0	0
Secondary Amplitude OP4	0	0	0	0	0	0
Release Rate OP1	0	0	0	0	0	0
Release Rate OP2	0	0	0	0	0	0
Release Rate OP3	0	0	0	0	0	0
Release Rate OP4	0	0	0	0	0	0
AM Enable OP1	0	0	0	0	0	0
AM Enable OP2	0	0	0	0	0	0
AM Enable OP3	0	0	0	0	0	0
AM Enable OP4	0	0	0	0	0	0
	Ch1	Ch2	Ch3	Ch4	Ch5	Ch6

Global Parameters	
LFO Enable	0
LFO Speed	0
DAC Enable	0
DAC Data	0

Channel 1 Bank	1
Channel 2 Bank	1
Channel 3 Bank	1
Channel 4 Bank	1
Channel 5 Bank	1
Channel 6 Bank	1

Figure 4: The genM YM2612 FM patch editor

Although the YM2612 possesses wide-ranging FM synthesis capabilities, there are techniques that can be used by the SEGA Mega Drive / Genesis Music Interface to extend the musical expression and sonic possibilities of the SEGA console. With regard to the YM2612 sound chip, there is a range of post-audio circuitry and components that distort and change the character of the sound. In particular, the digital to analogue converter tends to interact more with the other channels when set to a high, static value. The direct current produced by writing a value above 127 to the digital to analogue converter data register results in the remaining channels becoming more pronounced. This occurs even if the total output level of a given carrier oscillator or set of carrier oscillators is set low, as long as the oscillators themselves are active. These artefacts can be exploited creatively by presenting opportunities of making sounds not considered inherent to the YM2612 IC itself, such as an approximation of a low-pass filter.

As was pointed out earlier, the sixth channel of the YM2612 can act as a basic digital to analogue converter via a straightforward process, as follows. A DAC register value of zero will set the amplitude to the most negative excursion, whilst a DAC register value of two hundred and fifty-five represents the most positive excursion. Thus, audio samples can be played back from read-only memory by reading through a block of data and then, on a byte-by-byte basis, performing timed register updates. The SEGA Mega Drive has no specific hardware infrastructure intended for dealing with the timing operations of sample playback (for example, an audio buffer), and therefore all timing that deals with writing samples to the DAC register must be dealt with by programming the console.

However, by setting but not changing the value of the DAC register the sixth channel simply outputs a direct current (DC) signal. This direct current has some interesting interactions with the output of the other FM channels. For example, the DC signal appears to act as a filter that distorts the outputs of the five remaining FM synthesis channels. This provides unexpected textures arising from the slow rectification of the resultant signals, and an increase in noise from the power supply as the value approaches zero. A pop or click-like artefact can also be heard when jumping from a DAC value of zero to a relatively high value. Figure 5 and Figure 6 compare different settings of the

DAC and how this has an effect on a basic sine waveform.

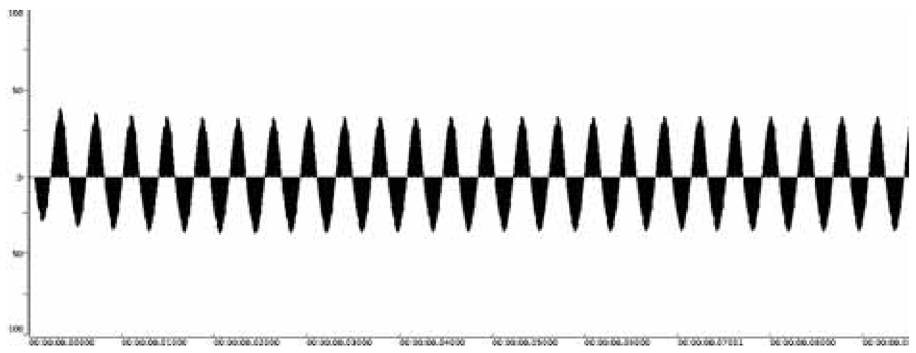


Figure 5: Waveform of a single-operator sinewave when the DAC is at a value of 0.

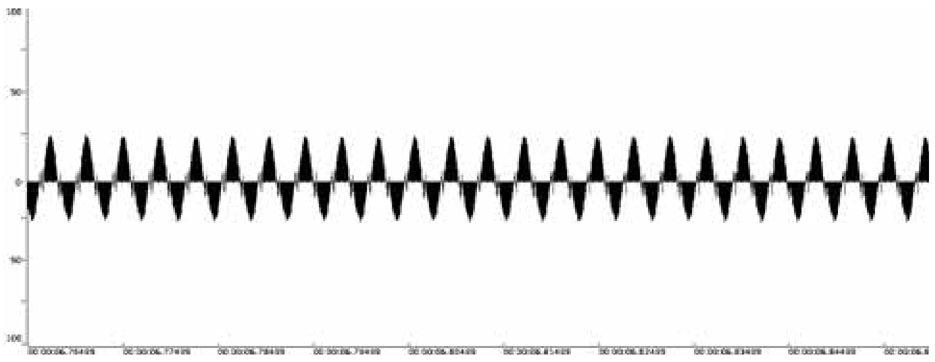


Figure 6: Waveform of a single-operator sinewave when the DAC is at a value of 80.

The three essential differences between these two waveforms result in significant sonic discrepancies. The waveform in Figure 6 is lower in amplitude, and grounding hum from the power supply can be heard with more prominence. There are additional, non-smooth portions of the waveform present whenever the zero axis is crossed. These portions of the waveform would account for what sounds like a filter frequency being moved. The clipped shape of the sine wave would account for any distortion that may be present.

The inclusion of post-audio circuitry unique to the SEGA Mega Drive gives the YM2612 its unique sound. The DAC can be used not only to play back samples, but also to synthesise sounds that are difficult or impossible to accurately generate with an FM chip, such as pseudo-random noise or a triangle waveform. The latter are achieved by the SEGA Mega Drive / Genesis Music Interface microcontroller calculating and then streaming digital samples, first to the 68K processor, and then on to the YM2612 chip. Thus, the user is able to explore sounds that would not be normally possible with that sound chip.

Summary

The SEGA Mega Drive console – being a staple of the 16-bit gaming era – has had both of its internal soundchips made accessible via the application of the Interface Framework described here. The two sound chips in question, the SN76489 and the YM2612, employ different sound generation techniques. Through mapping the control of sonic parameters and extending the functions of the sound chips, the YM2612 and SN76489 are transformed into useable musical devices.

The submitted creative work *Antia* (in particular the section 'Etcetera' on page 3 of the timeline) demonstrates the high level of control over the YM2612 afforded by the Interface Framework. *Antia* also showcases extended techniques, such as sample playback (see 'Circles', page 3), timbral morphing and filter emulation ('The Mountain Is On Fire!', page 1). *Office Yoga Made Easy* was composed entirely for the SEGA Nomad (the portable SEGA Genesis / Mega Drive equivalent), and explores FM synthesis techniques, rhythmic phases and the role of inherent noise in chipmusic.

SEGA Master System Music Interface

SEGA Master System Console Overview and Background

According to Jeff Bogumil, the 1986 SEGA Master System was SEGA's 'answer' to Nintendo's 8-bit NES (Nintendo Entertainment System), both of which were competing in the post-crash videogame marketplace of the mid 1980s. [23] Although not as universally popular as the NES, the SEGA Master System enjoyed moderate success in Europe, Australasia and Brazil.

The SEGA Master System included a number of interesting peripherals, such as the Light Phaser – a gun controller based on a weapon in the manga series *Trillion* – and rudimentary 3-dimensional glasses. The system was designed around the 8-bit Zilog Z80 microprocessor, which is also an integral feature of products as diverse as scientific calculators (for example the TI series by Texas Instruments[24]) and various MP3 players.[25] Additionally, the Z80 performs a variety of functions, including key scanning in analogue synthesisers such as the Sequential Circuits Prophet 5. [26]

Sound Capabilities of the SEGA Console

The SEGA Master System (SMS) features two distinctly different sound chips. The primary SN76489 chip is standard for every SEGA Master System,[27] while the secondary,

23 Jeff Bogumil. 'SEGA FAQ'. May 1999. http://www.cs.colostate.edu/~dzubera/FAQs/sms.faq (accessed 23 March 2010).
24 Phillip Parker. *Calculators: Webster's Quotations, Facts and Phrases* (San Diego: Icon Group International, 2008), 460.
25 Alexis Sepchat, Simon Descarpentries, Nicolas Monmarché and Mohamed Slimane. 'MP3 Players and Audio Games: An Alternative to Portable Video Games Consoles for Visually Impaired Players', in *Computers Helping People with Special Needs: 11th International Conference, ICCHP 2008*. Linz, Austria, July 9-11, 2008 (Berlin: Springer Berlin, 2008).
26 Paul Théberge. *Any Sound You Can Imagine: Making Music / Consuming Technology* (Middletown: Wesleyan University Press, 1997), 59.
27 Karen Collins. *Game Sound: An Introduction to the History, Theory, and Practice of Video Game Music and Sound Design* (Cambridge: MIT Press, 2008), 15.

less common YM2413 sound chip was included only on some Japanese models.[28] Although the SEGA Master System does not contain an actual, discrete SN76489 sound chip, the properties of the chip are incorporated into the custom visual display processor of the SMS.

The SN76489 is made up of three square wave channels and one pseudo-random noise channel. Each of the square wave channels of the SN76489 behaves identically, with neither the waveform type nor the duty cycle being adjustable. By playing the three channels together at different octaves with different volumes, it is possible to generate timbres that are impossible to produce using a single voice. This is a basic form of additive synthesis, albeit using square waves as opposed to the more traditional sine waves.

A 10-bit data register sets the frequency of each square wave generator, a relationship that is governed by the equation *Frequency = Clock / 32 * Data* (where *Frequency* is the output frequency, *Clock* is the master clock of the video game console [3546893 Hz for a PAL console, 3579545 Hz for an NTSC console] and *Data* is the contents of the frequency register). The lowest possible frequency for an NTSC console is approximately 109 Hz, which restricts sonic and therefore creative choices. By contrast, the highest possible frequency is an altogether more artistically satisfying 111800 Hz.

The majority of oscillators that derive their frequency from the division of a master clock signal feature an inverse relationship between frequency output and pitch resolution. The SN76489 has a relatively low pitch resolution in the two highest audible octaves, which is the product of a frequency register that yields only 1023 discrete frequencies. Figure 7 shows the relationship between SN76489 data and perceived pitch in the form of MIDI note values.

28 Phillip Parker. *Chips: Webster's Quotations, Facts and Phrases*, 313.

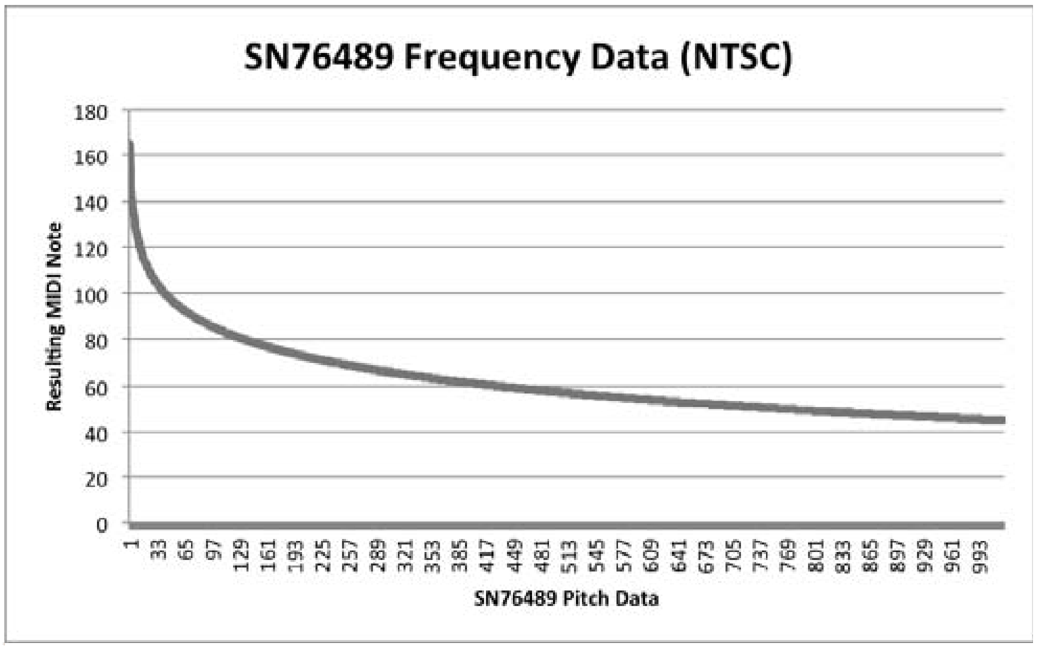


Figure 7: NTSC SN76489 frequency range represented as MIDI values

The pseudo-random noise channel of the SN76489 features eight modes, each producing a different pitch / timbre combination. The generator can be set to either pitched or non-pitched mode, with each containing four frequency settings – high, medium, low and externally clocked. The frequency of the noise channel in the externally-clocked mode mirrors the pitch of the channel 3 oscillator, albeit three octaves lower. Thus, the noise channel is able to generate frequencies that are far below 108.4Hz. However, while producing low frequencies using this method extends the pitch range of the SMS, it also restricts the polyphony as the third square wave channel effectively becomes unusable.

In terms of volume, the SN76489 features sixteen steps of non-linear amplitude control for each channel. While this does not greatly impact upon amplitude envelopes, it does negatively affect the playback of samples, as the positive and negative excursions of the waveform will not be balanced in terms of resolution. Figure 8 displays the graphing of the volume points available to the SN76489 when mapped against a linear amplitude scale.

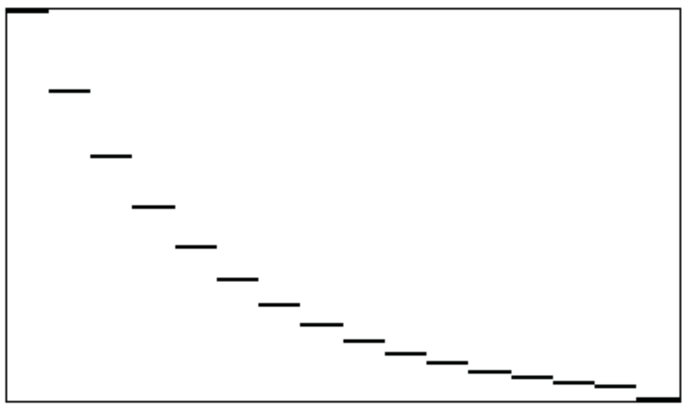


Figure 8: SN76489 volume data points mapped on a linear scale

In addition to the primary SN76489 sound chip, the SEGA Master System also features a YM2413 sound chip. The FM synthesis-based YM2413 is at the lower end of the price range of Yamaha FM sound chips. Although the YM2413 is found only in a number of earlier Japanese SEGA Master System models, reproduction circuits can be purchased adding FM functionality to consoles from any region.

Technically, the YM2413 offers a wider variety of timbres and greater number of individual voices in comparison with the SN76489. The YM2413 is made up of a single modulator / single carrier FM synthesis engine with an attack, sustain and release type envelope. The YM2413 makes use of a restrictive architecture where each channel can be set to one of sixteen instrument presets. Besides the first of such settings (instrument zero), all settings that deal with FM sound synthesis are predetermined. Upon recalling a given setting, all parameters are fixed in that particular configuration until the next setting is recalled. These preset instruments are shown in Table 8, below.

Preset Number	**Instrument Name**	**Preset Number**	**Instrument Name**
1	Violin	9	Horn
2	Guitar	10	Synthesiser
3	Piano	11	Harpsichord
4	Flute	12	Vibraphone
5	Clarinet	13	Synthesiser bass
6	Oboe	14	Acoustic bass
7	Trumpet	15	Electric guitar
8	Organ		

Table 8: YM2413 instrumentation presets.

The custom instrument (preset number zero) gives the user control over all parameters, allowing the YM2413 to create complex waveforms that can be transformed at will. Although this allows for extensive sonic control, the YM2413 is restrictive in that instrument zero is 'shared' between all channels; any changes made to instrument zero will affect all channels that are currently set to that instrument. Although the fifteen other instrumental presets cover a range of sonic archetypes, the sound chip is capable of a wider variety of timbres – as is demonstrated in the sonic morphologies of the creative work *Dynasty* (see Appendix B). This is in contrast with the more complex Yamaha FM chips, where there is complete control over every synthesis parameter of every voice. In terms of volume, each YM2413 channel provides sixteen discrete amplitude steps, where the least significant volume bit represents a change in 3dB, whilst every subsequent bit represents a doubling in dB from the previous bit.

Current Capacity for Music-making on the Console

Plogue Chipsounds offers an excellent emulation of the SN76489 sound chip that can be controlled via a range of popular software packages. Another method for creating SEGA

Master System music is using MOD2PSG, a music tracker software package for the Windows platform whose purpose is to sequence music for the SN76489 sound chip.[29] The MOD2PSG software, which shares the limitations found in most tracking environments, also inherits the basic limitations of the SN76489 chip itself. The data file from the MOD2PSG music package can be converted into a format that can be played back on SEGA Master System hardware. However, this method is limited to non-real time use of the SMS. The SEGA Master System Music Interface described below is intended to facilitate the use of extended sonic and compositional techniques, as well as real time performance – both of which are capabilities lacking in the MOD2PSG. Furthermore, the interface opens up the system for use with many popular, modern music packages.

The SEGA Master System Music Interface: Concept, Design and Capabilities

The SEGA Master System Music Interface has been created by applying the Interface Framework to the SEGA console, and consists of a microcontroller, software for the microcontroller and software for the video game console. The microcontroller interfaces with the SEGA Master System console via the player two controller port, allowing for sufficient data speed and complete control over the sound output. The 7 relevant data pins that partially make up the second controller connection can be read by the Z80 microcomputer on ports $DC and $DD.

The protocol to transfer information is straightforward, as only 8 bits of information need to be passed on to the SN76489 in order to update a channel, in contrast with more complex sound chips. Four pins act as a data nibble, with an additional pin acting as a high / low nibble flag. Another pin acts as a write flag, indicating that there is valid data present

29 Martin Konrad. MOD2PSG. Software for Windows OS. 2007. (KonTechs Limited). http://www.kontechs.de/product?name=mod2psg (accessed 16 March 2010).

at the data pins. More information regarding the specifics of the protocol can be found on the accompanying data disc (Appendix E) in the location *Major Case Studies/SEGA Master System/Source Code*.

The YM2413 FM chip is more complex than the SN76489, requiring 5 bits for the register address and 8 bits for the data to be sent to the address. The YM2413 is accessed using a sequence set in place by the SN76489 transfer routines, with an additional pin indicating the presence of FM data for the sound chip. The sequence is as follows:

1) The FM indication pin is set to a high state and kept there for the remainder of the YM2413 data transfer.
2) The address is then written to the SEGA Master System in the same manner that an entire byte is written to the SN76489. As the FM indicator pin is high, the Z80 microcomputer does not send the data to the SN76489. Instead, the microcomputer essentially waits for the data portion of the FM address and data pair.
3) Subsequently, the data byte of the pair is written to the SEGA Master System as two separate nibbles, which are then sent to the YM2413 address specified previously.

Refer to the accompanying data disc (Appendix E) at Major Case Studies/SEGA Master System/Source Code for the microprocessor and microcontroller source code.

The SEGA Master System Music Interface pushes the SN76489 and the YM2413 beyond their intended capabilities. The SN76489 can be extended by executing sample playback, amplitude and frequency modulation, direct waveform synthesis and a simple method of additive synthesis, as well as using the noise channel to output pitches. Although the YM2413 cannot be extended to the same degree, the instrumentation restrictions referred to previously can be compensated for.

The SEGA Master System Music Interface allows the end user to write data to the console significantly quicker than software emulations such as the MOD2PSG music package. As a result, unique sonic techniques can be explored such as simple vibrato / frequency modulation, simple tremolo / amplitude modulation, more complex use of the noise channel and sample playback.

Frequency and amplitude modulation are achieved using the same technique. In both cases, a continuous loop on the interface microcontroller writes information to the Z80, whereby either the control of frequency or volume is modulated over time. The amount and speed of the modulation in are controlled MIDI CC messages, with the resulting effect ranging from a mild vibrato or tremolo to more complex and distorted waveforms than those of which the SN76489 is inherently capable. The SEGA Master System Music Interface allows the square waveforms of the SN76489 to be organised into tightly controlled phase relationships, the nature of which generate a huge variety of unexpected timbres. This is achieved by first combining two voices that are tuned to the same frequency. One of the voices is then detuned for a brief period of time before being retuned to the same frequency. The extent of the detuning determines the phase shift amount; the longer the detuning period or the larger the detuning amount, the larger the phase shift, and with that, the greater the timbral shift. This method can be easily programmed and executed, and there are negligible audible artefacts resulting from the detuning. A range of timbres can be generated via this method, as seen in the sonogram in Figure 9. In this figure, the x-axis represents time whilst the y-axis represents frequency. The more vibrant colours indicate increased energy for a given frequency, whilst the cooler colours indicate an absence of energy for a given frequency.

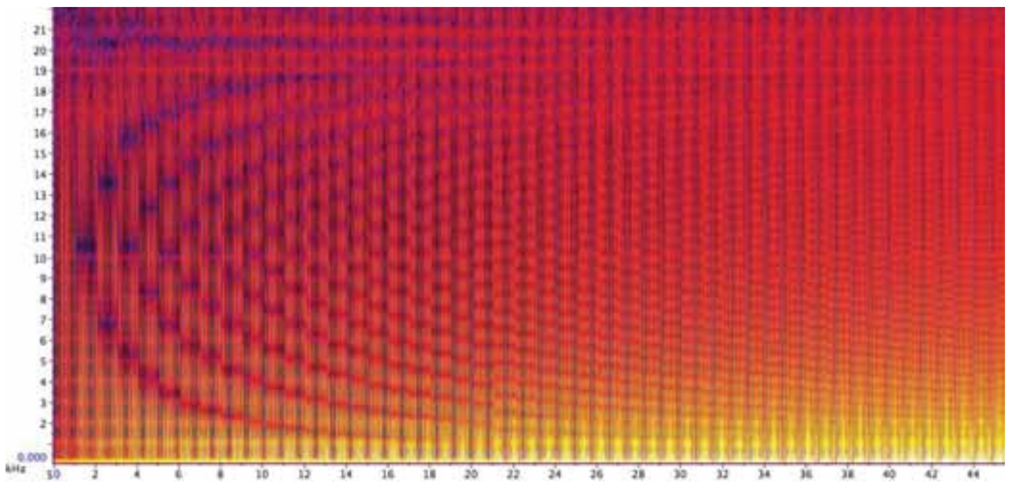


Figure 9: Sonogram of phase-based timbres

The noise channel can also be extended sonically using the Interface, via a series of rapid note-on and note-off events, simultaneously generating both a pitched bass line and a percussive noise drum line. This form of continuous MIDI data is suitably handled by an arpeggiator plugin within a host music environment.

Sample playback is possible despite the non-linear amplitude output of the SN76489, though samples are distorted and low in volume. Sample playback is achieved by reading through an array of bytes stored in the memory of the Interface microcontroller and writing each of these bytes to the SMS as volume data points. Each byte represents a single sample that, when played in order, forms a digital audio waveform.

The YM2413 can also be extended, albeit only in attempting to overcome the limitations of the instrumentation settings. As previously described, instrument zero is the only preset setting where the composer has control over the sound that is produced. However, all channels set to instrument zero will sound identical, resulting in a limited amount of low-level sonic control within a polyphonic context. There are two ways in which this limitation can be overcome using the Interface – instrument change envelopes and instrument change modulation.

The concept of an instrument change envelope is that each note-on event automatically also triggers an instrument change event shortly thereafter. Two predefined instrument settings are chosen – setting A and setting B. Instrument setting A is before the note-on event and instrument B is after. By changing the settings whilst the chip is currently synthesising sound for a given voice it is possible to produce timbres that are not strictly the result of a predetermined instrument setting, but rather a combination of the two. This process is summarised in Figure 10.

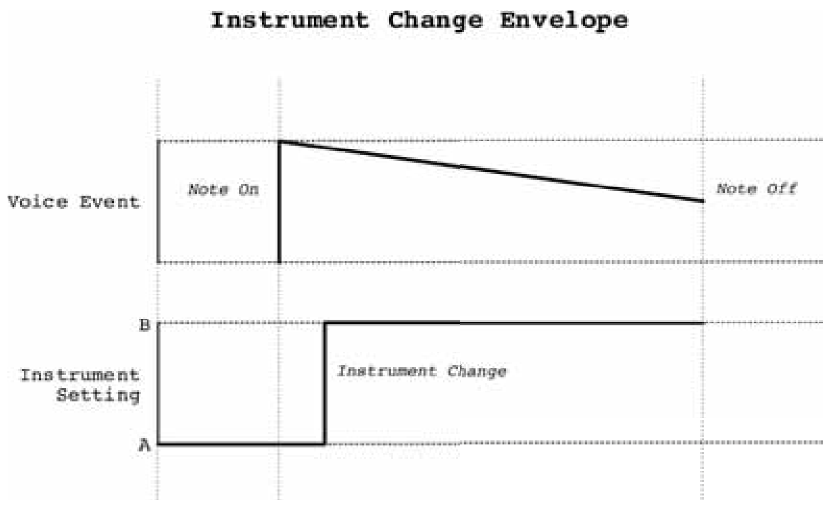


Figure 10: YM2413 instrument change envelope.

It is also possible to modulate the instrument settings of a given voice. When a note-on event is received, a software-based control oscillator is activated, which is then used to modulate between two instrument settings, thus creating more complex sounds than available using the presets individually. This concept is summarised in Figure 11.

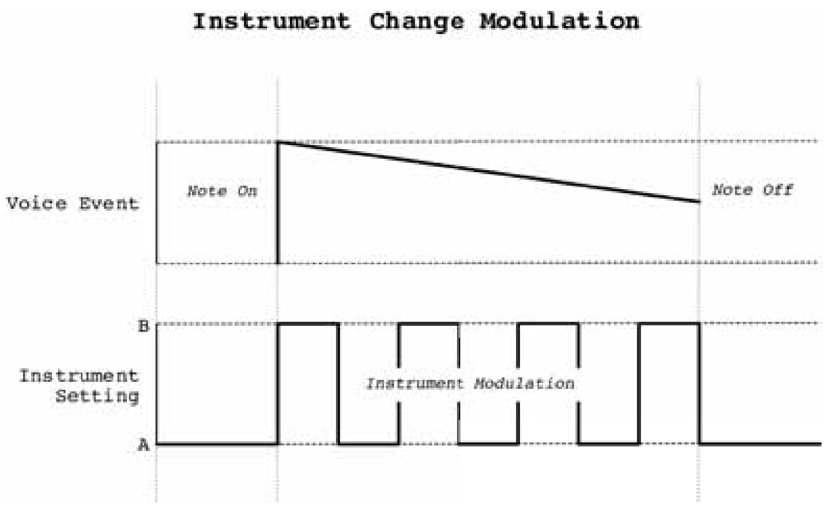


Figure 11: Instrument change modulation

In both cases, a simple Max/MSP patch allows the user to set the instrument preset numbers and customise the speed and onset of the program change modulation or envelope that handles the mapping of notes to MIDI control-change messages.

With up to nine pitched channels, the YM2413 chip is a suitable candidate for experimentation with an artificial, data-driven reverberation effect. This reverb or delay effect is facilitated by the Interface in a straightforward manner, by sending similar musical material to all channels. Although the same pitch and timbre is sent to all channels, the volume data is reduced and a short delay is introduced for every subsequent channel. This process results in a cascade of note data that can mimic a delay or reverberation effect, the exact nature of which is determined by the delay time increment and the volume scaling across the nine channels. A Max/MSP patch has been created that performs the task of delaying, changing and routing the note data of a single, monophonic line. Titled YM2413VERB, the patch features a virtual MIDI input and a MIDI output that should be connected to the SEGA Master System Music Interface, and a range of controls relating to the reverberation or delay effect.

All of the techniques described above allow the user of the SEGA Master System Music Interface to create unique SN76489 and YM2413 music using the SEGA hardware.

Summary

The classic 8-bit SEGA Master System console contains the SN76489 programmable sound generator chip in all models, while only some Japanese models are packaged with the YM2413 FM sound chip (which is nonetheless compatible with all models regardless of origin).

Both sound chips have been made easily accessible to the musician and composer via the SEGA Master System Music Interface. Not only have all inherent parameters and features of the SN76489 and YM2413 been made controllable via music data commands, but extended techniques allow for the exploration of sounds that are unique and not associated with either sound chip under standard operation.

The artistic capabilities of the Interface can be heard in two of the creative works included in Appendix B. *Error Repeat* features the Interface-controlled SN76489 alongside the Nintendo Entertainment System's sound chip. Here, at page 1, lines 2 and 3 of the timeline, the SN76489 provides melodic and rhythmic elements. *Dynasty* explores the limits of the YM2413 sound chip and the wide range of textures that can be created under the control of the Music Interface. Timbral changes, for example, are indicated on the first two lines of page 1 of the timeline.

Atari 2600 Music Interface

Atari 2600 Console Overview and Background

The Atari 2600, released in 1977, is the oldest video game console examined in this work. Unlike the dedicated, single-game consoles that preceded it, the Atari 2600 popularised the use of removable software cartridges and microprocessors in videogames. The arrival of interchangeable, inexpensive cartridges reduced overall costs to consumers, extended the marketing life of the console, and opened up a whole new platform for software development.

The Atari 2600 console was compatible with a variety of what were rapidly expanding peripherals, which included keypads, steering wheels and paddles. A peripheral named the SuperCharger enabled the console to produce higher resolution graphics using a 64KB RAM expansion pack.[30] Games that were programmed to be used with the SuperCharger could be stored as data waveforms on standard audio cassettes. A unique feature of the SuperCharger was an audio line input to which a user connects the headphone output of a cassette player that subsequently loads the software. This set up can be modified to enable homebrew developers to easily test programs on an Atari 2600 machine.

The Atari 2600 has limited audio and video capabilities, both of which are handled by the custom Television Interface Adapter (TIA) chip. The video game console lacks video RAM, and as such can only draw a single line of video output at a time. Thus, programming visuals for the Atari 2600 can become complex, as the process of drawing graphics to a display must be timed with great accuracy.[31] All code must be synchronised

30 Joe Grand, Frank Thornton and Albert Yarusso. *Game Console Hacking: Have Fun While Voiding Your Warranty* (St. Louis: Elsevier Inc, 2004), 393.

31 Nick Montfort and Ian Bogost. *Racing the Beam: The Atari Video Computer System* (Cambridge: MIT Press, 2009), 28.

with the television refresh rate.

Although writing software for the Atari 2600 video game console is notoriously difficult, the classic console has enjoyed something of a revival in recent years. Many modern platforms such as Windows, Mac OS X, Linux, Nintendo Wii, X-Box and X-Box 360, PlayStation 1 and 2, Apple iPhone and Nintendo DS are capable of emulating the Atari 2600. Furthermore, the Atari 2600 homebrew scene is vibrant with multiple titles being released each year.

The development of user-programmed software is made possible by a strong, competitive homebrew culture that thrives on clear, well-written technical documentation. AtariAge[32] is an important website in this context and features technical documentation and community interaction in addition to an online store. The AtariAge store plays an important role for Atari 2600 development, stocking all of the physical materials needed to produce a homebrew title, as well as software titles that have been programmed by members of the community. As such, the store serves a double function; it provides developers the materials needed to develop new titles, and these are in turn made available to the broader Atari 2600 video game community. The AtariAge store also stocks a limited number of chipmusic compact disc releases, including albums that feature the Atari 2600 alongside other instrumentation.

Sound Capabilities of the Console

The TIA chip containing two basic audio oscillators handles the sound generation of the Atari 2600. These basic oscillators are musically constrained by an exceptionally low frequency resolution and limited polyphony. Although the chip can create a range of timbres, all the default waveforms produced by the TIA are 1-bit. As such, an oscillator

32 Albert Yarusso. 'Atari Age - Have You Played Atari Today?' http://www.atariage.com/ (accessed 15 March 2010).

can only be in a high or a low state (on or off) at any point in time. Different timbres are produced by predefined combinations of high and low voltages throughout the period of a waveform. It is these timbres that help give the console a unique sound that other chips cannot reproduce.

A 4-bit timbre register and a 5-bit frequency register together determine the waveform and pitch of a given oscillator.[33] Although there are 16 possible choices for timbre, and each choice might be expected to yield a unique set of 32 pitches, limitations of design have caused an overlap between timbre choices and pitch sets. As such, there are 9 possible timbre types and 7 pitch ranges, as summarised in Table 9, below. Note that all frequency ranges are given for PAL-based TIA chips and, although the pitch ranges for their NTSC counterparts are similar, they are not precisely the same.[34]

Distortion	**Frequency Range (Hz)**	**Description**
0	-	-
1	65 - 2080	Somewhat distorted 1
2	2.1 - 67	Periodic noise with pitched elements 1
3	2.1 - 67	Periodic noise with pitched elements 2
4	485.7 - 15600	Rectangle wave 1
5	485.7 - 15600	Rectangle wave 1
6	31.5 - 1006	Somewhat distorted 2
7	31.5 - 1006	Somewhat distorted 3
8	1.9 - 61.1	Periodic noise
9	31.5 - 1006	Somewhat distorted 3
10	31.5 - 1006	Rectangle wave
11	-	-
12	162.5 - 5200	Rectangle wave
13	162.5 - 5200	Rectangle wave
14	10.5 - 335.5	Rectangle wave
15	10.5 - 335.5	Somewhat distorted 4

Table 9: Atari 2600 distortion / timbre summary

33 Atari technical documentation generally refers to the *timbre* register as a distortion register, and although this may be a mathematically correct term when dealing with linear feedback shift registers, the term timbre was chosen here as it is musically more relevant to the current study.

34 The difference in tuning between the NTSC and PAL Atari 2600 models stems from the different crystal oscillator speeds used in order to generate timing for the respective video signals.

Timbre types 0 and 11 are effectively silent, although the output of an oscillator set to either timbre type will carry a voltage proportional to the relevant volume register value. This capability is discussed below. Timbre type 15 is arguably the most recognisable Atari 2600 waveform because of its iconic use in commercially successful games such as *Pitfall!*

Each timbre type limits a given oscillator to a set of 32 pitches that are musically unrelated. The graph in Figure 12 shows the pitch set possible when using timbre type 1. The horizontal axis represents the data value of the 5-bit frequency register from 0 to 31, while the vertical axis represents the resulting frequency of that value as a MIDI pitch. Note the inverse relationship between the lowering of frequency and the increased pitch resolution, as well as the large pitch range that is covered by the 32 discrete frequencies.

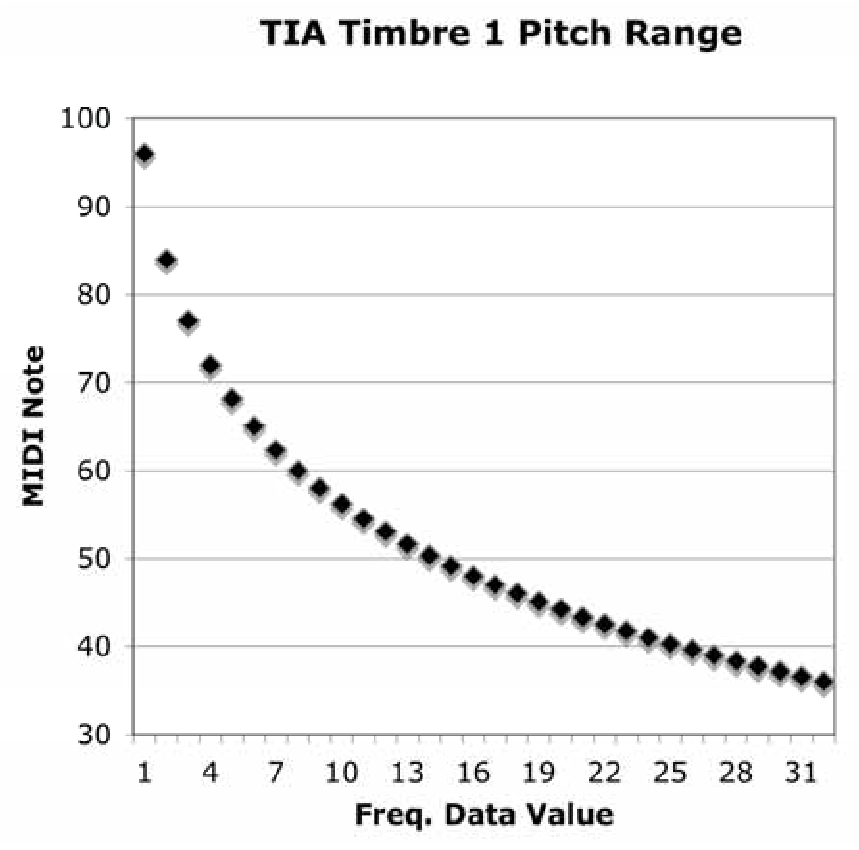


Figure 12: All possible frequencies of TIA timbre type 1

In addition to the timbre and pitch registers of a given channel, a 4-bit volume register is used to scale the amplitude of the oscillator to 1 of 16 possible volume levels.

This volume scaling is achieved by first storing the 4 bits that make up the volume register in an array of latches. Each latch can retain 1 bit of information, which is sent as either a high or low voltage signal to one of four NOR gates. Also connected to an input of each NOR gate is the 1-bit signal from one of the two oscillators. A resistor network is connected to the output of the NOR gates, acting as a digital to analogue converter (DAC). This DAC takes the 4-bit signal representing both the volume data and the frequency of the oscillator, and outputs a representative analogue voltage. The relationship between the volume register data, NOR gates, oscillator signal and DAC is shown is Figure 13.[35]

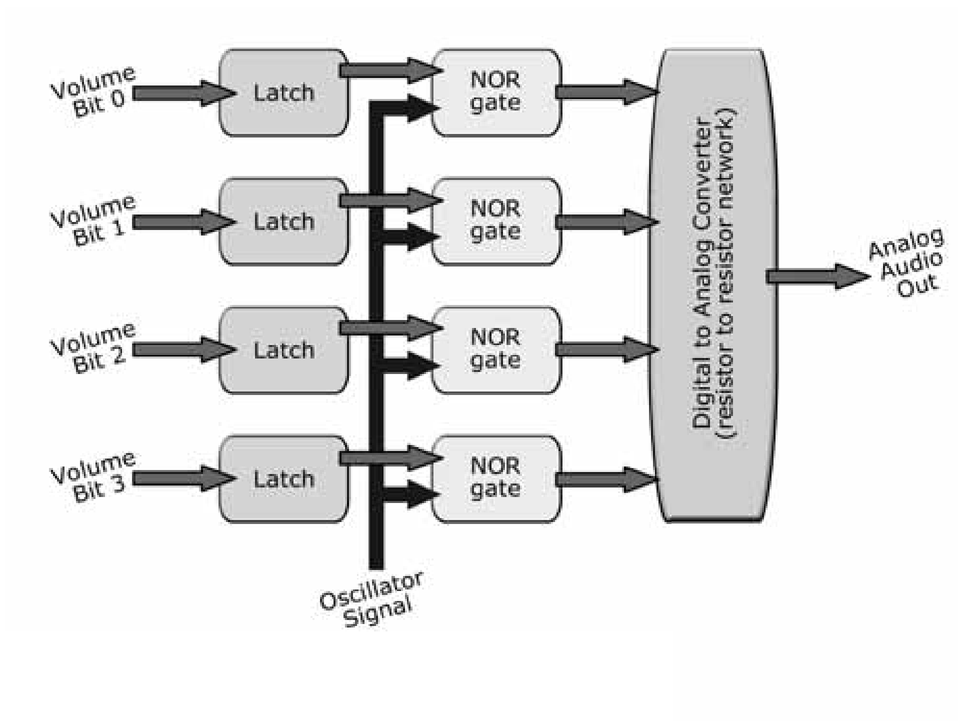


Figure 13: TIA oscillator output stage.

The use of 1-bit waveforms with a 4-bit volume control as described above is common to many programmable sound generator chips, including the SN76489 and the SAA-1099. The Atari 2600 is able to play back digital audio samples using volume data – this process is described in the 'Extended Techniques' section of this case study.

35 Atari Corporation. TIA Schematic, page 4. http://www.atariage.com/2600/archives/schematics_tia/TIA_1A_400dpi_4.zip (accessed 15 March 2010).

The Atari 2600 is built around the MOS technology 6507 microprocessor, which is a cost-reduced version of the 6502 microprocessor.[36] The TIA chip handles all video and audio functions of the Atari 2600 by receiving data from the 6507 microcomputer and routing appropriate signals to a television set via an RF connection. In order to achieve a clear audio signal, it is necessary to create a direct audio output from the TIA chip, rather than simply demodulating the RF output.

The NTSC and PAL versions of the TIA chip differ; the former offers an individual pin for each oscillator, while the latter internally combines both audio oscillators into a single output pin. A direct audio output can be added to an Atari 2600 console as follows. Bearing in mind that some level of electrical protection is advised, a suitable decoupling circuit is shown in Figure 14 and Figure 15. This will protect the Atari 2600 from power supply issues such as voltage spikes.

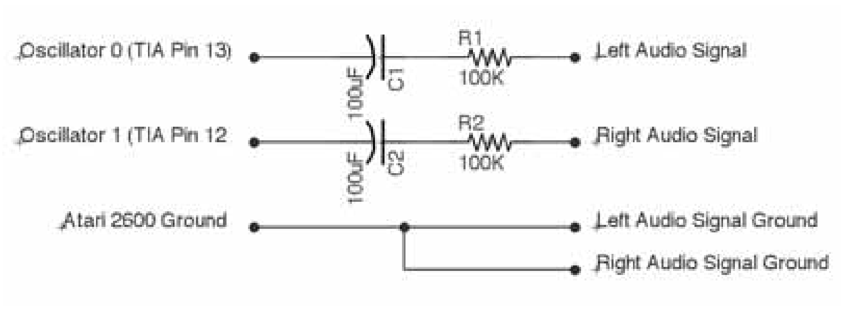


Figure 14: Direct audio output for an NTSC-type TIA chip

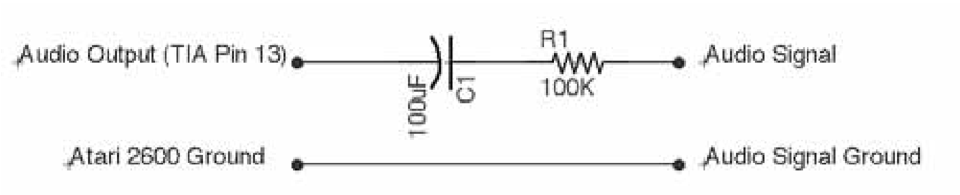


Figure 15: Direct audio output for a PAL-type TIA chip

36 Nick Montfort and Ian Bogost. *Racing the Beam: The Atari Video Computer System* (Cambridge: MIT Press, 2009), 12.

The component values given in the above figures do not need to be exact. The capacitors should be above 47 uF and can be up to 200 uF, while the resistors can be as low as 10k Ω or 20k Ω. The value of a resistor will change the signal level – the higher the resistance, the lower the signal level. However, a lower signal level is not necessarily a negative attribute, as the audio signal level directly from the TIA chip is quite high and may cause clipping in some recording systems. Both the resistors and capacitors in an NTSC modification should be of identical values, otherwise the audio oscillators will be of different output levels. The addition of the audio output to the Atari 2600 makes the console more suitable for use in the computer music studio and for music composition.

Current Capacity for Music-making on the Console

Apart from programming directly in 6507 assembly language or in the Atari BASIC (Batari) language, there are three ways that music can be written for the Atari 2600, all of which have been developed by artist, musician and programmer Paul Slocum. These three methods include using Slocum's Synthcart[37] and Loopcart[38] software, as well as his Atari 2600 Music Sequencer Kit.[39] Although each method serves a specific purpose, none of them provide a user-friendly means of retaining complete audio control over the TIA.

Slocum's best known method for making music using the Atari 2600 is his popular Synthcart cartridge, whereby the console is transformed into an instrument that can be used for live performance. Two twelve-button keypads are used for input and allow the user to control pitches and select percussive loops. Furthermore, parameters such as basic attack,

---

37 Paul Slocum. Synthcart. Software for Atari 2600. Published by AtariAge.com. 2002. http://www.qotile.net/synth.html (accessed 15 March 2010).

38 Paul Slocum. Loopcart. Software for Atari 2600. 2004. http://www.qotile.net/loopcart.html (accessed 16 March 2010).

39 Paul Slocum. 'Atari 2600 Sequencer Kit'. http://www.qotile.net/sequencer.html (accessed 16 March 2010).

sustain and release envelopes, tuning mode (major, minor or atonal) can be manipulated, and arpeggio speed can be set in addition to the global tempo. In the case of controlling pitches, Slocum has done an excellent job of managing voices and creating a relatively playable instrument, considering the limited range of pitches that can be produced using the console. However, the cartridge does not allow the performer / composer to use all of the possible pitches that the Atari 2600 can produce.

Additionally, the global tempo of the Synthcart is limited to specific values. This is due to the fact that Synthcart displays visual information and, as previously stated, the Atari 2600 requires significant processing power to create a video signal. As a result, the mechanisms used to control musical timing can only be updated at limited chronological intervals, resulting in a set of rigid tempi. These predefined tempo values are accurate and stable, albeit unusual. The NTSC version of Synthcart can only play at the following tempi: 224.71 BPM, 179.77 BPM, 149.81 BPM, 128.41 BPM, 112.36 BPM, 99.87 BPM, 89.88 BPM, 81.72 BPM, 74.90 BPM and 69.14BPM, with the PAL version featuring a similar yet equally limited tempo set. There are large incremental steps within these values, making the music sound rough and disjointed as opposed to gradual and smooth.

Loopcart, Slocum's unfinished program for the Atari 2600, features a simple compositional environment whereby the music output of the console can be sequenced into a series of patterns. Loopcart does not have a tempo control in its current form, with the tempo default being approximately 120 BPM. Furthermore, it is not possible to save any pre-sequenced musical material using the Loopcart. Considering these limitations, the Loopcart is more suited for experimentation rather than a full exploration of the sonic and musical capabilities of the TIA.

In addition to Loopcart and Synthcart, Slocum has released an Atari 2600 Sequencer Kit, which is essentially an environment that can be used to compile music for the 2600 using a custom sound engine derived from 6507 assembly language. Slocum states that the aim of the kit is to promote the composition of music for the console, especially in the context of homebrew games:

> You can use it to write backing beats or tracks for your music, or develop music for 2600 homebrew games. The driver uses a very small amount of 2600 resources so it can easily be placed into game code. There is a growing demand for 2600 homebrew music so it would be great to have more 2600 music programmers.[40]

Although the Sequencer Kit provides the composer with a higher degree of control over the Atari 2600 sound than either Loopcart or Synthcart, the composition process is limited by the inherent exclusion of real-time control, algorithmic processes and complex musical structures. These limitations are overcome by creating an Atari 2600 Music Interface, giving the user complete control over the sonic capabilities of the TIA chip.

The Atari 2600 Music Interface: Concept, Design and Capabilities

The Interface Framework has been applied to the Atari console, resulting in the Atari 2600 Music Interface. As stated previously, the TIA chip handles all video and audio output, and as such is responsible for sound generation. The TIA contains 6 registers that control the sound output, and writing data to any of these will have an immediate effect. The six registers are shown in Table 10, below.

40 Paul Slocum. 'Atari 2600 Sequencer Kit'. http://www.qotile.net/sequencer.html (accessed 16 March 2010).

Register	**Function**	**Length (Bits)**
AUDC0	Timbre control for voice 0	4
AUDC1	Timbre control for voice 1	4
AUDF0	Frequency control for voice 0	5
AUDF1	Frequency control for voice 1	5
AUDV0	Volume control for voice 0	4
AUDV1	Volume control for voice 1	4

Table 10: Atari 2600 TIA sound register.

These registers can be accessed via the 6507 microcomputer. Data is sent to the Atari 2600 via a predetermined protocol in which one portion of a byte acts as an address area, while another portion acts as data. Due to the small length and number of TIA audio registers, only a single transfer of information is required to completely update the sound output of the Atari 2600. The upper three bits of the protocol represent the address portion, whilst the lower five bits represent the data that is to be written to the appropriate register, as determined by the address.

Unlike the SEGA Master System Music Interface or SEGA Mega Drive Music Interface, the Atari 2600 Music Interface makes use of both controller ports as opposed to a single port. The joystick direction information for both controller ports is used to transmit data, and can be found at Atari 2600 RAM location SWCHA. Table 11 shows the relationship between SWCHA, the physical Atari 2600 pins and the relevant Atari 2600 Music Interface protocol data function.

SWCHA Bit	**Atari 2600 Pin**	**Protocol Function**
Bit 7	P2 RIGHT	Address Bit 2
Bit 6	P2 LEFT	Address Bit 1
Bit 5	P2 DOWN	Address Bit 0
Bit 4	P2 UP	Data Bit 4
Bit 3	P1 RIGHT	Data Bit 3
Bit 2	P1 LEFT	Data Bit 2
Bit 1	P1 DOWN	Data Bit 1
Bit 0	P1 UP	Data Bit 0

Table 11: Atari 2600 direction assignment in RAM

The player 1 button, which appears at location INPT4, acts as a write signifier whereby a low state activates the write cycle and indicates to the 6507 that relevant data is currently present at the controller port. The three address bits form eight individual address states. The address states zero to five correspond to the sound registers show in Table 12.

Address Data	**Sound Register**
000	AUDV0 (Volume control for oscillator 0)
001	AUDV1 (Volume control for oscillator 1)
010	AUDF0 (Frequency control for oscillator 0)
011	AUDF1 (Frequency control for oscillator 1)
100	AUDC0 (Distortion control for oscillator 0)
101	AUDC1 (Distortion control for oscillator 1)

Table 12: Atari 2600 protocol addresses

The circuit design of the interfacing hardware for the Atari 2600 Music Interface differs to the SEGA Master System Music Interface and the SEGA Mega Drive Music Interface. While the latter two share the same MIDI input to single 9-pin connector circuit, the Atari 2600 Music Interface uses a MIDI input to dual 9-pin connector circuit, as shown in Figure 16. Due to the use of a dual port scheme, the Atari 2600 Music Interface is able to transfer information quickly and easily to the Atari 2600. As a result, the inherent sonic capabilities of the TIA chip can be augmented using extended techniques.

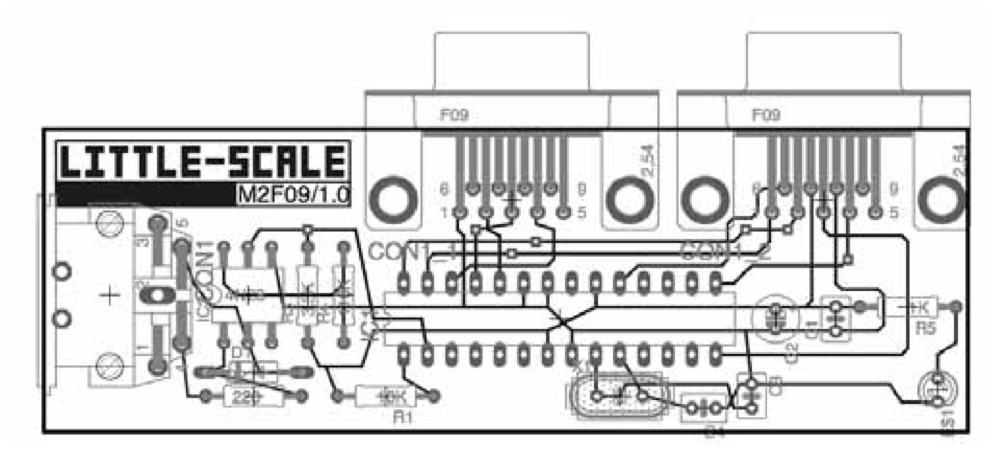


Figure 16: Dual 9 pin D-subminiature MIDI Interface Hardware PCB

Although the rudimentary sound capabilities of the Atari 2600 limit aspects of musical composition – especially in the area of pitch selection – the basic synthesis section of the TIA chip does allow for certain extended techniques. The TIA distortion types 0 and 11 are effectively silent in terms of the waveform produced. In essence, these distortion types are always in a high voltage state, where the voltage is proportionate to the volume register data. As a result, the volume register can be continuously updated at brief intervals with new data, thus allowing the channel to be used for the playback of audio samples and software-based synthesis waveforms. Although programmers recognised this technique during the so-called 'golden age' of the Atari 2600 video game console (1977-82), audio samples were not often used, owing to the lack of computational power and memory storage space.

In relation to the former, playing back samples whilst displaying graphics is computationally demanding given the 6507 microcomputer architecture. For sample playback to occur, the microcomputer needs to read each individual sample from the cartridge read-only memory, sending the data to the sound portion of the TIA integrated circuit. This process needs to be performed on a sample-by-sample basis with the correct timing, as the TIA lacks an audio buffer that is able to load samples in sequence.

Commercial Atari 2600 cartridges are either 4KB or 8KB in size, thereby lacking the memory space to store data such as audio samples.

However, the Atari 2600 Music Interface releases the 6507 microcomputer from the burden of executing any code other than interpreting data from the music interface hardware, sample playback can be easily achieved. The data containing the samples is stored in the microcontroller program (and not the Atari 2600 program), and is read back and sent to the Atari 2600, and subsequently the TIA, in real time whenever a request is made to trigger a given sample.

As the sample playback is a function of volume modulation, the depth of the samples is limited to 4 bits. When reading back the sample data stored in the microcontroller ROM, every odd-numbered sample is read from the low nibble of a data byte, whereas every even-numbered sample is read from the high nibble of the corresponding data byte. The sample rate upon playback is approximately 8,000 Hz. The Atari 2600 Music Interface microcontroller has 27KB of free space after incorporating all functionality needed to communicate with the Atari 2600 and to control the audio oscillators of the TIA chip. The audio samples reside in this free space. A selection of audio material has been processed, converted and included in the ‘sketch’ that makes up the microcontroller program.

Sample Number	Name
1	707 electronic drum machine kick
2	707 electronic drum machine snare
3	707 electronic drum machine hi-hat (closed)
4	707 electronic drum machine hi-hat (open)
5	707 electronic drum machine clap
6	707 electronic drum machine tom 1
7	707 electronic drum machine tom 2
8	707 electronic drum machine snare rim shot
9	electronic kick
10	electronic snare
11	electronic cow bell *
12	BRK sample
13	circuit bent guitar sample *
14	chamber orchestra hit *
15	YM2413 kick drum
16	YM2413 snare drum
17	YM2413 high hat
18	YM2413 tom
19	YM2413 cymbal
20	piano chord *
21	ukulele chord 1 *
22	ukulele chord 2 *
23	ukulele chord 3 *
24	ukulele chord 4 *

Table 13: Atari 2600 sample set

The aim of the choice of samples shown in Table 13 is to maximise the variety of expression and musicality available when using such a small sample set. Samples marked with an asterisk are pitched, allowing the end user to incorporate additional sonic elements within melody lines, thereby creating interest and helping to de-cliché the sound palette of the Atari 2600 console.

Rhythmic samples allow the end user to add a variety of drum-like sounds, instead of relying entirely on the noise waveforms of the TIA chip for percussive parts. Furthermore, these samples can be extended and varied by being pitch-shifted with two MIDI Continuous Controller parameters. One MIDI CC shifts the sample play back rate down in small increments by adding 'ticks' between sample updates, whereby a set number of ticks is counted depending on the CC value. The other CC parameter can shift the pitch up by effectively omitting a set number of samples for every sample that is played back. A similar approach to the method of sample playback as described above can also be applied to software-based synthesis.

By extending the sonic capabilities of the TIA chip inside the Atari 2600, the Atari 2600 Music Interface derived from the Interface Framework allows the user to explore composition with the console using a greater degree of freedom and expression than previously possible.

Summary

The Atari 2600 video game console is one of the oldest video game consoles still readily available, and features unique yet limited sound capabilities via the custom Television Interface Adaptor (TIA) chip. Through the development of the Atari 2600 Music Interface described here, all audio aspects of the TIA are brought under the complete control of the end-user. Additionally, sample playback and related manipulation greatly extend the expression possible with the Atari 2600 video game console, creating new possibilities for musical composition.

The full force of the Atari 2600 Music Interface can be heard in the creative work *Antia*, wherein the sound capabilities of the TIA chip are pushed to their limits, resulting in a work of intense textures, complex rhythms and unique sounds. *Antia* demonstrates techniques such as extensive Atari 2600 sample manipulation (as heard in the section 'Chermside', timeline, page 2) and full control over the synthesis processes of the TIA chip (as heard in 'Exit Wounds', timeline page 1). Furthermore, the Atari 2600 Music Interface used in conjunction with the Atari console provides extensive percussive material, in the form of sample playback, as is demonstrated in *Office Yoga Made Easy* ('We Are Eating Our Children', timeline, page 1).

GCE / Milton Bradley Vectrex Music Interface

GCE / Milton Bradley Vectrex Console Overview and Background

Released in 1982, the GCE / Milton Bradley Vectrex holds a unique place in video game history as it is the only home console to feature vector-based graphics. The aim of the Vectrex was to recreate a home experience of the vector-based arcade games from the late 1970s and early 1980s. The console was designed around the 8-bit 6809 microcomputer which, having been designed to approach 16-bit performance at a minimum cost to the user, was viewed as 'one of the most powerful 8-bit processors to come to market'.[41] The Vectrex was designed as a stand-alone unit and is complete with its own vector display monitor featuring a unique vertical format.[42] Yet, for all its innovation, the Vectrex fell victim to the 1984 video game crash, which saw it reduced to an abject market failure that lasted only two years.[43]

The vector graphics employed by the Vectrex rely on the analogue positioning of one or more cathode ray tube beams in order to draw a line between any two points on a screen. Images can be rescaled to any size by being stored as a map of co-ordinates, disposed as the beginning and end points of the lines.[44] The use of vector-based graphics contrasts with raster, pixel-based displays used in all other home video game and computer systems. Raster graphics do not allow the creation of true diagonal lines or the enlarging of graphic sprites without loss of quality.[45] Although the graphics hardware of the Vectrex does not have a direct impact on the sonic capabilities of the console, the screen emits a

41 Andrew Staugaard. *6809 Microcomputer Programming and Interfacing, With Experiments* (Indianapolis: Howard W. Sans, 1981), 7.
42 Mark Wolf. 'Imaging Technologies', in *The Medium of the Video Game* (Austin: University of Texas Press, 2002), 19 - 23.
43 Chris Kohler. *Retro Gaming Hacks* (Sebastopol: O'Reilly Publishing, 2005), 37.
44 Nell Dale and John Lewis. 'Vector Representation of Graphics' in *Computer Science Illuminated.* 4th ed. (Boston: Jones and Bartlett Publishers, 2010), 81 - 82.
45 Steven Molnar and Henry Fuchs. 'Advanced Raster Graphics Architecture', in *Computer Graphics: Principles and Practice*. James D. Foley (ed). 2nd ed. (Reading: Addison-Wesley Professional, 1995), 855 - 922.

buzzing sound whose amplitude and frequency is dependent on the number, size and positioning of the displayed vectors.

Sound Capabilities of the Console

Unlike most video game consoles, the Vectrex has no direct audio line output and was designed with an internal speaker only. As such, the sound of the Vectrex is always coloured by the physical space in which the console is used. This colouration can be bypassed by modifying a Vectrex console so as to have a direct audio output.

The Vectrex features an AY-3-8912 sound chip – a programmable sound generator made up of three channels. The AY-3-8912 is a sonically identical variant of the AY-3-8910 sound chip, lacking one of the input / output ports found on the latter. Each channel of the AY-3-8912 can be set to a pitch-enabled mode for which the frequency is selectable. The sound chip also features a noise-enabled mode whereby a global pseudo-random noise source is routed to the output of a given channel. The frequency of the noise source is selectable via a global register. These two modes are not mutually exclusive, as a channel can simultaneously output both pitch and noise material. The frequency resolution and frequency range of the AY-3-8912 are beyond that of comparable sound chips, such as the Texas Instrument SN76489. However, as with the SN76489, the oscillators of the AY-3-8912 cannot be adjusted in terms of duty cycle and waveform type. Furthermore, the sound chip features a single amplitude envelope generator that can be used to modulate the volume of a channel.[46] Although all of the features of the AY-3-8912 can, in theory, be used for music composition with the Vectrex, original Vectrex games tended to feature sparse soundtracks that under-utilised the capabilities of the sound chip.

46 General Instruments Semiconductor Group. AY38910, AY38912, AY38913 Datasheet. http://www.msxarchive.nl/pub/msx/mirrors/hanso/datasheets/chipsay38910.pdf (accessed16 March 2010).

Current Capacity for Music-making on the Console

There is currently no readily accessible method of creating music using the Vectrex console that affords complete control over the AY-3-8912 sound chip. As with many other video game consoles, it is possible to write music by programming a series of register updates in assembly language. However, this method of music composition is not flexible or time efficient. The Vectrex includes inherent BIOS functions that allow the play back of music without having to write data to the sound chip on a register-by-register basis.[47] Although these BIOS functions serve to minimise the time and effort required to compose music, they also simplify the level of control exerted by the programmer over the sound chip. Unique to the Vectrex BIOS is a set of thirteen short tunes that can be used by a programmer to add pre-written music to a game.[48]

Melody Master,[49] an original Vectrex title aimed at children, focuses on music composition. A light pen peripheral allows the user to draw notes onto a vector musical stave and then play back phrases. However, this software is limited in the phrase length, pitch range, note length, time signature and tempi that it can deliver, and as such not suitable for more complex music composition. Additionally, *Melody Master* is monophonic despite the sound chip being able to play back up to three notes simultaneously.

The Vectrex Music Interface, developed by applying the Interface Framework to the video game console, gives the user complete control over the AY-3-8912 sound chip, circumventing the simplicity of software such as *Melody Master* and the rigid complexity

47 Chris Salomon. 'An Introduction to Vectrex Programming: Appendix A: Overview of Functional Classes: 12: Sound'. http://www.playvectrex.com/designit/chrissalo/appendixa.htm#12 (accessed16 March 2010).
48 Chris Salomon. 'An Introduction to Vectrex Programming: Sound Playing'. http://www.playvectrex.com/designit/chrissalo/soundplaying.htm (accessed16 March 2010).
49 GCE Corporation. *Melody Master*. Software for Vectrex. 1983.

of assembly language music composition.

GCE / Milton Bradley Vectrex Music Interface: Concept, Design and Capabilities

The Vectrex Music Interface has been developed by applying the Interface Framework to the Vectrex video game console. The Vectrex console features two controller ports, both of which are used by the Vectrex Music Interface for data transfer. The standard Vectrex controller has one analogue joystick and four buttons. In regard to the Vectrex Music Interface, the button inputs on the controller port are used as the conduit for data. The button state, or voltage, of these eight pins is read by the AY-3-8912, which sends the relevant data to the 6522 VIA peripheral chip. The 6522 VIA chip, in turn, sends this data to the 6809 microcomputer, where it can be processed.[50] Once the incoming data has been read and processed by the 6809, the new values are sent from the 6809 microcomputer through the 6522 VIA peripheral chip to the AY-3-8912 sound chip. This flow of information is summarised in Figure 17.

50 Chris Salomon. ‘Appendix C: The 6522A Chip From The Vectrex Perspective’. http://www.playvectrex.com/designit/chrissalo/via3.htm (16 March 2010).

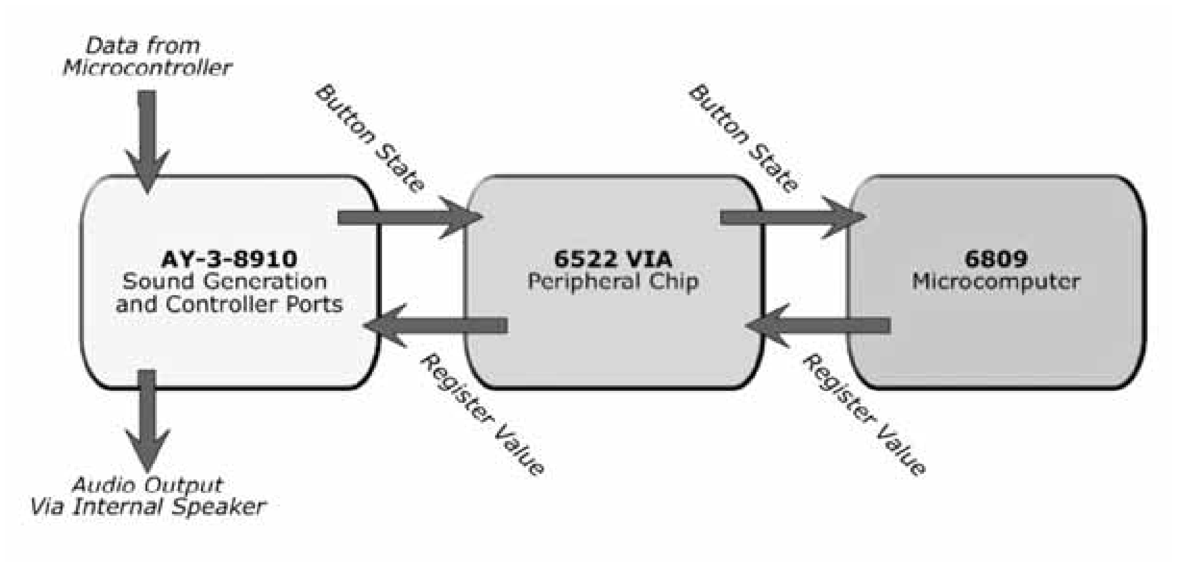


Figure 17: Information flow in the Vectrex interface

The Vectrex Music Interface protocol used to transfer data to the Vectrex is similar to the SEGA Mega Drive Music Interface protocol (see the SEGA Mega Drive / Genesis Music Interface case study, pp. 35-46). The eight button states of the Vectrex are read using a subroutine, after which the data is available in a predefined RAM location (specifically $C80F). Thus, the data pins function together within a protocol so as to transfer data to the Vectrex, as shown in Table 14.

Bit	**Button**	**Protocol Function**
7	P2, B4	write signifier
6	P2, B3	address / data type
5	P2, B2	high / low nibble
4	P2, B1	(unused)
3	P1, B4	data bit 3
2	P1, B3	data bit 2
1	P1, B2	data bit 1
0	P1, B1	data bit 0

Table 14: Vectrex button inputs at location $C80F and protocol functions.

The lowest 4 bits are used as the data nibble, corresponding to either the target address or the data that should be written to the target address. Bit 7 is the write signifier, which indicates to the 6809 that new data is present. Bit 6 is the type signifier, which indicates whether the data nibble is of the address type or the data type. Bit 5 is the nibble signifier, which indicates whether the data nibble is high or low. The write cycle consists of first writing the low data nibble, then the high data nibble, and then finally the target address nibble. As the AY-3-8912 has 16 registers, only one write event is required to update the target address. The 6809 updates the AY-3-8912 once these three write events have been received by the Vectrex. This process requires approximately one millisecond to complete; a length of time that is considerably longer than the other case studies. As a result, the inherent sound capabilities of the AY-3-8912 can only be extended by a limited amount.

In contrast to the three other console interfaces developed through the current study, the Vectrex Music Interface is unable to play back samples, as the process of transferring data from the microcontroller to the Vectrex is time consuming and results in an unworkably low sample rate. Nonetheless, the limited rate of information transfer is sufficient for the real-time synthesis of an additional noise source from the microcontroller. This additional noise source overcomes the limitations associated with the internal noise source of the AY-3-8912, where the frequency of the noise waveform is determined by a global parameter rather than on a channel-by-channel basis. By supplying an additional noise source via the Vectrex music interface, multiple channels of the AY-3-8912 are able to simultaneously output noise at different frequencies.

Although the Vectrex Music Interface only allows for a single extended technique in this instance, it nevertheless facilitates complete control over all inherent functions of the AY-3-8912 sound chip. As such, it presents an unprecedented level of musical expression and potential for composition using the console.

Summary

The unique GCE / Milton Bradley Vectrex vector-based video game console had a short market life, but remains a sought-after collector's item, with many homebrew titles continuing to be programmed for it. The Interface Framework when applied to the Vectrex results in a Vectrex Music Interface that yields complete control over the AY-3-8912 sound chip. The application of the Interface additionally introduces a custom noise mode, which in turn overcomes the limitations caused by the noise source intrinsic to the sound chip.

Selected Discrete Sound Integrated Circuit Interface Designs

In addition to exploring interfacing with fully-fledged video game consoles, the current study has also applied the Interface Framework to discrete sound chips. A discrete sound chip is a standalone integrated circuit package and may be associated with one or more video game consoles and arcade game machines.

The transformation of discrete sound chips into useable musical devices represents an opportunity to explore sonic palettes previously unavailable to the electronic musician and composer. This chapter discusses musical Interfaces that have been built by the author for a number of discrete sound chips, including the General Instruments SP0256-AL2, the Texas Instruments SN76489, the Atari POKEY and the Philips SAA-1099. Aspects of the Interface Framework are applied to the control of the above sound chips. MIDI data is received and interpreted by the microcontroller, which then sends control information to the sound chip in question. Thus, the microcontroller plays the role of the microcomputer.

Extending Functionality

Functionally, there are three main advantages in using discrete sound chips by contrast to their in-console counterparts – controlling the frequency of a master clock signal, modulating a master clock signal, and complete control over the sound output stage of the circuitry.

The first advantage of utilising discrete sound chips is that the master clock signal used for a given sound chip can be of a frequency value not stated nor recommended by the manufacturer. The master clock signal is a simple squarewave whose frequency controls all timing procedures within a discrete sound chip. These procedures include functions such as the frequency of oscillators, and the speed of envelope generators. The frequency of the master clock can be altered or replaced with a variable frequency oscillator, thereby affecting the frequency range and resolution of pitch. Halving the speed of the clock for the discrete SN76489 is an example of manipulating a master clock signal for a specific outcome. The pitch range of the SN76489 is shifted down by an octave, allowing the end user to use notes of a lower register than previously possible.

In addition to simply replacing or changing the master clock signal, a second oscillator can be used as a modulator – a process resulting in interesting timbral changes in the output of the sound chip. Additionally, a discrete sound chip allows the designer / builder to exercise complete control over the output stage of the sound chip. Although this may seem of little consequence for the end user, it should be noted that the circuitry appearing after a sound chip in a video game console may be poorly designed or manufactured – an issue that is overcome by the use of discrete sound chips.

The designer has complete control over the audio output stage of the sound chip in the case of discrete sound chips. This creates the possibility of a customised and unique sound, such as is the case with the SP0256-AL2. The schematic for this discrete sound chip includes a passive, single-pole filter, the purpose of which is to eradicate unwanted digital noise. By omitting the filter, the sound of the SP0256-AL2 becomes more crunchy – it takes on a more 'digital' character.

A similar example of the potential offered by control of the audio output stage can be heard in the audio output of the SAA-1099 programmable sound generator. In the technical documentation supplied by Philips, no values are given for the resistor and capacitor that form the output filter. The lack of consistency in the production of these components results in a similar inconsistency in their actual values, the net result of this being a wide variety of timbral outputs.

General Instruments SP0256-AL2

The SP0256-AL, being a speech synthesis sound chip, is able to generate all of what it claims are the 59 phonemes that make up the English language.[51] Essentially, the SP0256-AL2 is an extension of previous speech sound chips made by General Instruments, the main difference being that earlier chips required external memory for the storage of phoneme data. As a microcomputer requires resource-intensive transfer operations when accessing such external memory, designers were able to offload this burden by combining the phoneme data with the sound synthesis core of the SP0256-AL2.[52] The application of a microcontroller to the sound chip allows for phonemes to be strung together to create interesting musical events, including unexpected percussive sounds and unusual timbres. The schematics and Arduino microcontroller code for connecting to and controlling the SP0256-AL2 can be found on the data disc (Appendix E) in Minor Case Studies/SP0256-al2/.

51 General Instruments Semiconductor Group. SP0256-A Datasheet. http://lib.store.yahoo.net/lib/e-clec-tech/spo256.pdf (accessed 22 March 2010).

52 General Instruments Semiconductor Group. SP0256-A Datasheet.

Texas Instruments SN76489

The SN76489 sound chip is functionally identical to the sound generator of the SEGA Master System video game console, though the latter does not possess a discrete chip. A cloned version of the SN76489 is found within the SEGA visual display processor. The SN76489 is also found in other consoles, such as the ColecoVision.[53]

All basic sounds of the SEGA Master System can be reproduced with the discrete SN76489 Interface. However, by manipulating the master clock signal of the discrete chip, sounds can be generated that would be impossible using an unmodified SEGA Master System or ColecoVision video game console. The schematics and Arduino microcontroller code for connecting to and controlling the SN76489 so as to manipulate the master clock signal can be found on the data disc (Appendix E) in Minor Case Studies/sn76489/.

Atari POKEY

The Atari POKEY (an abbreviation for potentiometer and keyboard) integrated circuit was developed by Atari for the Atari 5200 video game system and the Atari 8-bit computer line. The POKEY sound chip was also used with a small number of Atari 7800 games. Interestingly, the POKEY was not included as part of the 7800 game console itself. Rather, the sound chip was integrated with the actual cartridge games that made use of the POKEY.[54]

The Atari POKEY is able to produce a range of timbres and noise patterns.[55] The POKEY has the ability to generate with a wide pitch range. This is due to the sound chip's

53 Karen Collins. *Game Sound: An Introduction to the History, Theory, and Practice of Video Game Music and Sound Design* (Cambridge: MIT Press, 2008), 24.

54 Joe Grand and Ryan Russell. *Hardware Hacking: Have Fun While Voiding Your Warranty* (St. Louis: Elsevier, 2004) 201 - 202.

55 Don Mahurin. 'Atari POKEY to Csound Conversion'. December 2006. http://pokey.openright.org/ (accessed 22 March 2010).

ability to combine the resources of two of its four voices for up to 16-bits of frequency resolution. A unique feature of the POKEY was the inclusion of two digital high pass filters, allowing for a range of tone colours.[56] The schematics and Arduino microcontroller code for accessing the POKEY can be found on the data disc (Appendix E) in Minor Case Studies/Pokey/.

Philips SAA-1099

The SAA-1099 by Phillips was used in the SAM Coupé. The discrete sound chip has six voices with six pitched oscillators, two envelope generators and two noise sources.[57] Although the chip is designed as a basic programmable sound generator, the frequency range is amongst the best of the PSGs of the era. This is due to the inclusion of an octave register as well as a frequency register, whereby the octave register sets the octave to one of 8 octaves. The frequency register sets the pitch identity, with a total of 256 discrete frequencies playable within a selected octave.[58] This musical approach to the mapping of pitch to frequency resulted in a sound chip whose frequency resolution is very even across a large pitch range. A unique aspect of the SAA-1099 is the excellent design of stereo output, whereby any voice can be routed to the left or right channel with an accuracy of 16 discrete volume steps per audio channel.

Although the timbre of the pitched oscillators of the SAA-1099 cannot be altered, multiple voices can be ganged together using sequencing software. Thus, a rudimentary form of additive synthesis can provide some variation in tone. The schematics and Arduino microcontroller code for connecting and controlling the SAA-1099 can be found on the

56 Chris Crawford, Lane Winner, Jim Cox, Amy Chen, Jim Dunion, Kathleen Pitta, Bob Fraser, and Gus Makrea. 'De Re Atari - Chapter 7'. Atari Archives: Software and Information. 1982. http://www.atariarchives.org/dere/chapt07.php (accessed 20 March 2010).

57 'Adding a Sound Chip'. http://www.onastick.clara.co.uk/sound.htm (accessed 22 March 2010).

58 Philips Corporation. Phillips SAA-1099 Complex Programmable Sound Generator Datasheet. http://velesoft.speccy.cz/samcoupe/saa1099/saa-1099_data_sheet.zip (accessed 22 March 2010).

data disc (Appendix E) in Minor Case Studies/SAA-1099/.

Conclusion: Future Directions

The aim of the current research has been to develop new methods for creating electronic music by exploiting obsolete technology. This goal has been reached both practically and on a theoretical level. An Interface Framework – here defined as an overarching concept for the direct control of sound chips found in obsolete video game consoles through the use of a microcontroller and relevant real time control – has been designed, refined and applied. The principle underpinning the Framework is the use of two software components (microcontroller and computer code) and one hardware component (the microcontroller itself) in order to exert absolute audio control with the view to creating new soundscapes and instruments. The Framework has been applied to a number of video game consoles, including the SEGA Mega Drive, SEGA Master System, GCE / Milton Bradley Vectrex and the Atari 2600. Furthermore, the Interface Framework has also been applied to a number of discrete sound chips, such as the General Instruments SP0256-AL2, the Texas Instruments SN76489, the Phillips SAA-1099 and the Atari POKEY.

The music interfaces developed through the various applications of the Framework have been deployed for artistic ends in the composition of four major creative works, totalling one hour of music. Additionally, audio material generated by manipulating four discrete sound chips for musical ends has been recorded and provided here. The latter material has been organised into a series of sample packs so as to afford the user access to the sounds created by the chips if that same user lacks the practical expertise to build the Interfaces.

The perceived benefit of the Interface Framework is to make new technologies available to the community in general, and the chipmusic community in particular. These technologies can be applied for immediate artistic ends, or they can constitute a point of departure for further explorations into the production of new instruments and sonic palettes arising from the retrospective exploration and re-purposing of obsolete technologies. A number of other consoles rarely used in chipmusic could be made more accessible via the application of the Interface Framework. These include the Commodore VIC-20, the Atari Jaguar, Atari 5200 and the SEGA Saturn. By opening up previously untapped resources, the Framework will ensure that the chipmusic scene remains vital well into the future.

Appendix A: Glossary of Terms

8-Bit Music: Colloquialism that is generally synonymous with the terms ‘chiptune’ and ‘chipmusic’ (depending on usage).

Additive Synthesis: The creation of complex sounds achieved by combining the waveforms of simple waveforms (usually at integer ratios of a fundamental frequency). In the context of this work, the term refers to an extended technique whereby additional timbres can be generated by combining a number of pulse waveforms at differing harmonic and amplitude relationships. *Examples include: as used via the SN76489 and the SAA-1099.*

Analogue Stick: In video gaming, the analogue stick is a special type of directional control that allows the console to measure not only the direction indicated by a player but how far in that direction a player is pressing. This creates the potential for a higher degree of correlation between a player’s action and a character’s resulting action. A digital stick only reads information based on direction rather than directional amount.

AY-3-8910: PSG Sound (and interfacing) chip found within a number of video game consoles and computers such as the GCE / Milton Bradley Vectrex. Comprising 3 channels, with 3 pitched pulse waves, 1 global noise source and 1 global volume envelope.

Bit: Smallest information element in digital computing. An area of memory, storage or representation that can hold one of two possible values, zero or one. Numbers commonly encountered that are expressed in bits or range of bits are 4-bit (16 possibilities), 7-bit (128 possibilities), 8-bit (256 possibilities) and 10-bit (1024 possibilities).

BIOS: Short for Basic Input / Output System, the BIOS of a computer system provides basic functionality between the various hardware components communicate with each other in order to function correctly. In the context of a video game console, the BIOS might contain pre-defined functions that allow the microcomputer to access other area of memory or to perform specific functions.

Byte: Common information element in digital computing. Comprised of eight bits. Commonly is used to represent eight one-bit states (8 x 0 or 1), two four-bit states (2 x 0 - 15) or a single eight-bit state (1 x 0 - 255).

Chipmusic: The composition of music using either (a) obsolete video game and computer music hardware or (b) stylistic elements of video games from the era of obsolete video game and computers (generally considered to be late 1970s to mid 1990s).

Chiptune: Two uses exist of this word. The first is synonymous with chipmusic, and indicates the movement as a whole. The second use is more specific and refers to the use of the composition of music via sound chip music data formats, or sound chip hardware (i.e. a purist approach).

Digital to Analogue Converter (DAC): An electronic component or sub-component that takes a digital input, such as a string of binaries numbers or a parallel input of high / low voltage combinations, and outputs a corresponding voltage. For example, an 8-bit (0 – 255) DAC with a range of 0 to 5V will output 0V if sent the number 0, ~2.5V if sent 127 and ~5V if sent 255.

Discrete Sound Chip: In the context of this work, this term refers to a sound chip that is an individual component and not part of a video game console. *Examples include the SAA-1099 and SP0256-AL2 sound chips.*

Flash Cart: In the context of this work, this term refers to an end-user rewriteable cartridge for a specific video game console. Flash carts are available for a range of consoles, including Nintendo Game Boy, SEGA Master System, SEGA Mega Drive etc.

Frequency Modulation (FM) Synthesis: The creation of complex sounds by changing the frequency of one or more simple waveforms based on the output one or more simple waveforms. The most commonly used waveform is the sine waveform. *Examples include the YM2413 and the YM2612 sound chips.*

Nibble: A 4-bit area of memory, storage or representation. Capable of representing sixteen discrete states. Two nibbles make up a byte.

NOR Gate: An electronic component or sub-component that has 2 digital inputs and one digital output. The state of the output depends on the state of the 2 inputs; the output will be high if and only if both inputs are low.

Microcontroller: A small, fully inclusive integrated circuit with dedicated, internal random access memory as well as read-only memory that performs pre-programmed tasks. Relevant platforms include Arduino, PixAxe and BasicStamp II.

POKEY: Discrete PSG sound (and interfacing) chip found within a range of Atari-related consoles and computers. Comprised of 4 oscillators that can be combined for higher frequency resolution and two digital filters.

Programmable Sound Generator (PSG): A sound chip that produces simple waveforms, most commonly pulse waveforms. Frequency is derived by the division of a master clock signal. *Examples include the SN76489 and the AY-3-8912 sound chips*.

Protocol: In the context of this work, this term refers to the method through which data is transferred from one computational system to another via electrical signals (and the associated, specific timing and significance thereof).

Pulse Waveform: In the context of this work, this term refers to a simple type of audio waveform whose output is either high or low. Commonly generated by a Programmable Sound Generator or similar circuit. The significant parameters of a pulse waveform are frequency, duty cycle and amplitude. *Examples include the waveforms produced by the SN76489 and the SAA-1099*.

Raster Graphics: A method for storing and displaying digital images using a (usually rectangular) grid of discrete points.

SAA-1099: Discrete PSG sound chip found within the SAM Coupé. Comprising 6 channels with 6 pitched pulse waves, 2 global noise sources, 2 global envelopes and a stereo audio output bus.

SN76489: PSG Sound chip found within a number of video game consoles and computers such as the SEGA Master System and SEGA Mega Drive. Comprising 3 channels and 1 noise source.

Sound Chip: In the context of this work, this term refers to an integrated circuit that functions to generate and output audio. Usually a part of (or associate with) a video game system. *Examples include: the YM2612 and AY-3-8912 sound chips.*

Source-Filter Synthesis: A method of synthesising speech whereby the vocal tract is emulated using a sound source (both pitched and unpitched) and a series of controllable filters.

SP0256-AL2: Discrete Speech sound chip for speech synthesis. Can reproduce all 59 phonemes of the English language. Based on the source-filter method of speech synthesis.

TIA: Sound (and video) chip found within the Atari 2600. Comprising 2 low-resolution oscillators with selectable timbre types.

Tracker: A type of music sequencing software that traditionally represents music using vertical structures made up of repeating blocks of data. Trackers often employ sample-based manipulation.

Vector Graphics: A method for storing and displaying images using a series of lines described by their beginning and end points.

YM2413: FM Sound chip found within the SEGA Master System. Comprising 9 FM pitched channels or 6 FM pitched channels and 5 FM percussion channels. 18 operators in total.

YM2612: FM Sound chip found within the SEGA Mega Drive. Comprising 6 FM channels with 4 operators per channel and a DAC mode.

Appendix B: Written Commentaries and Timelines Relating to the Creative Works

Overview

Broadly speaking, the creative works that follow can be characterised as having one of two functions: they either demonstrate the capabilities of the selected video game console as a solo musical instrument, or they demonstrate how the selected video game console can function alongside existing music technology.

The creative works are presented on the accompanying audio compact disc (Appendix D). The timelines for the creative works are available on the accompanying data disc (Appendix E) in the location Creative Works/ as well as being provided in hard copy in this appendix.

Creative Works Audio Compact Disc Track Listing

Please see the relevant written commentaries and timelines for more information regarding each work.

Title of Work: *Dynasty*
Instrumentation: SEGA Master System (YM2413 chip)
Duration: 12:59

Track Number	**Title**	**Duration**
1	*Dynasty* (Part 1)	5:34
2	*Dynasty* (Part 2)	7:26

Title of Work: ***Office Yoga Made Easy***
Instrumentation: SEGA Nomad and Atari 2600
Duration: 20:30

Track Number	**Title**	**Duration**
3	We Are Eating Our Children	4:18
4	Four-Wheel Drive Advertisement	2:00
5	Plus One	4:08
6	Self Replicating Technology	3:16
7	Latitude and Longitude	2:31
8	Microbiology	4:21

Title of Work: ***Antia***
Instrumentation: SEGA Mega Drive and Atari 2600
Duration: 11:30

Track Number	**Title**	**Duration**
9	The Mountain Is On Fire!	1:02
10	Cloaks And Daggers	1:57
11	Exit Wounds	1:24
12	Chermside	1:34
13	The Question	1:19
14	Etcetera	2:21
15	Circles	2:01

Title of Work: ***Error Repeat***
Instrumentation: SEGA Master System and Nintendo Entertainment System
Duration: 15:32

Track Number	**Title**	**Duration**
16	Come Back To Me	4:10
17	A Gentle Slope	1:59
18	The Merchant	4:08
19	Less Than	2:55
20	Campaign	2:23

Total Duration : 60:31

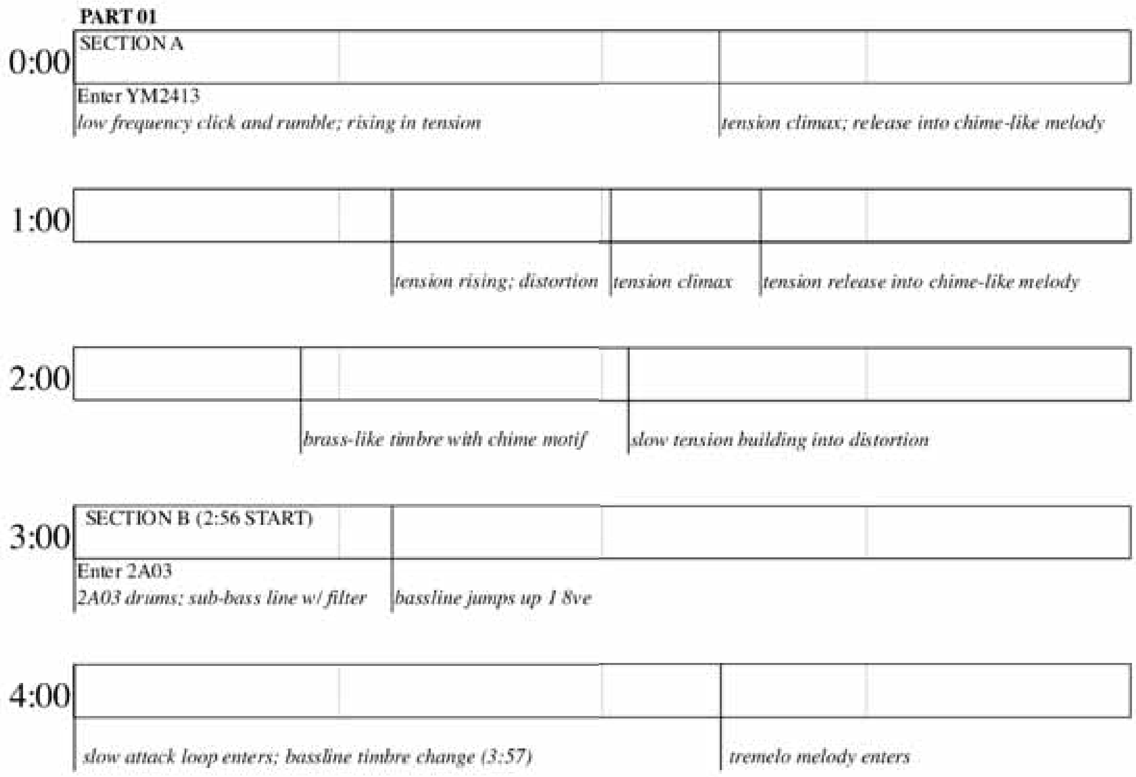
Written Commentary and Timeline: Dynasty (2008)

Dynasty, for SEGA Master System and Nintendo Entertainment System, explores the junction where minimalist and progressive chipmusic collide. *Dynasty* demonstrates the musical capabilities of the YM2413 sound chip opened up through the application of the Interface Framework. All pitched material is generated by the YM2413, while all percussive material is sourced from the Nintendo Entertainment System (making use of Chris Kann's MIDINES technology).

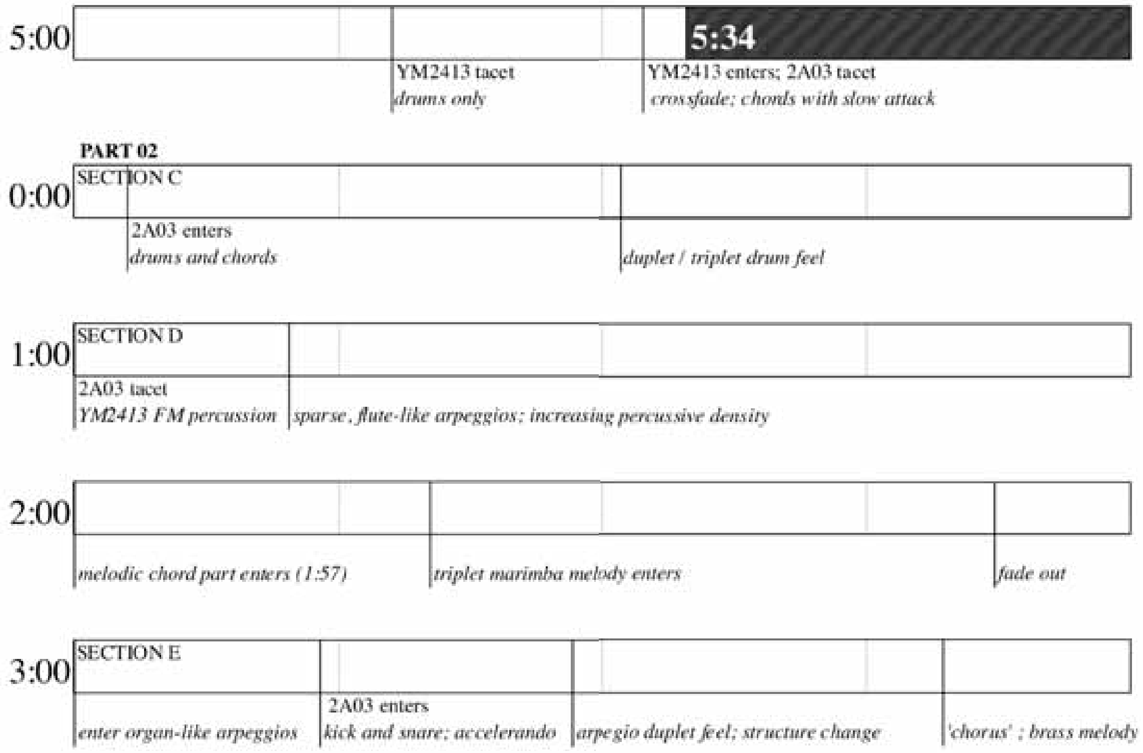
An important feature of this work is the full exploration of customised voicing using the YM2413 FM sound chip. Timbral morphologies are expressed by a complete command of the synthesis process, resulting in music that is at times soft, gentle and atmospheric whilst at others a torrent of energy. Rhythmic phasing and other compositional elements are heard throughout the work.

The timeline for this work is notated using a simplified graphical notation. Every horizontal row represents one minute of audio. Entry and exit points for instruments are shown. Structural divisions and musical highlights are marked on the timeline using text. In essence, *Dynasty* is in two parts (as marked on the timeline). The world premiere of this work was given at the 2009 Australasian Computer Music Conference as part of an electronic / electroacoustic music concert series of peer-reviewed creative works. Furthermore, *Dynasty* was released in 2009 on the progressive chipmusic label Pause Music and was performed during the 2009 Blip Festival in New York City, USA.

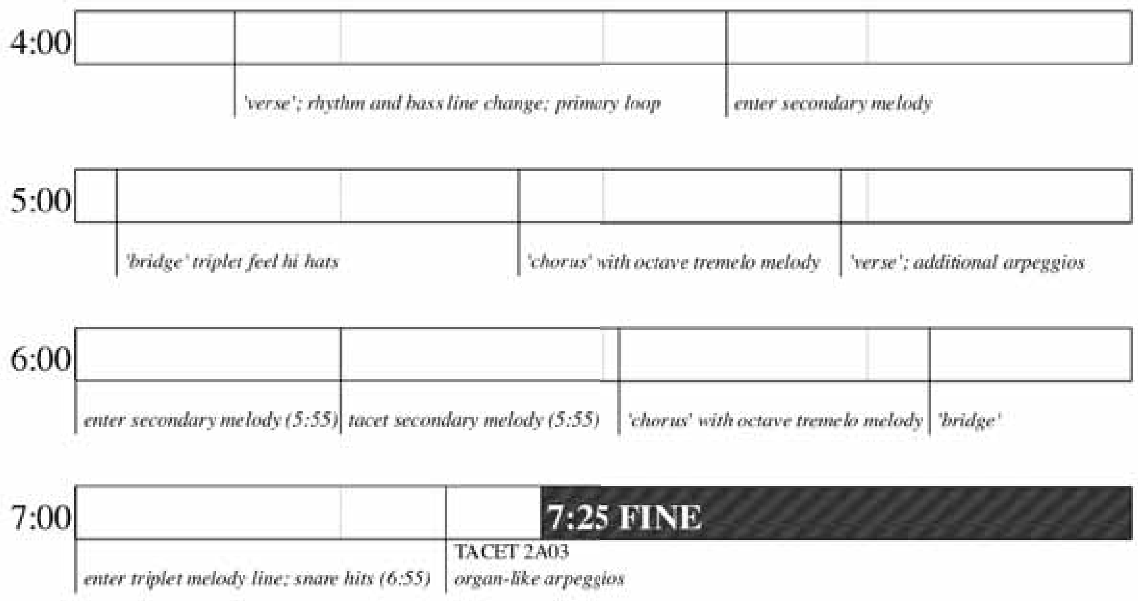
Dynasty - Page 1


PART 01
0:00
SECTION A
Enter YM2413
low frequency click and rumble; rising in tension
tension climax; release into chime-like melody
1:00
tension rising; distortion
tension climax
tension release into chime-like melody
2:00
brass-like timbre with chime motif
slow tension building into distortion
3:00
SECTION B (2:56 START)
Enter 2A03
2A03 drums; sub-bass line w/ filter
bassline jumps up 1 8ve
4:00
slow attack loop enters; bassline timbre change (3:57)
tremelo melody enters

Dynasty - Page 2


5:00
5:34
YM2413 tacet
drums only
YM2413 enters; 2A03 tacet
crossfade; chords with slow attack
PART 02
0:00
SECTION C
2A03 enters
drums and chords
duplet / triplet drum feel
1:00
SECTION D
2A03 tacet
YM2413 FM percussion
sparse, flute-like arpeggios; increasing percussive density
2:00
melodic chord part enters (1:57)
triplet marimba melody enters
fade out
3:00
SECTION E
enter organ-like arpeggios
2A03 enters
kick and snare; accelerando
arpegio duplet feel; structure change
'chorus' ; brass melody

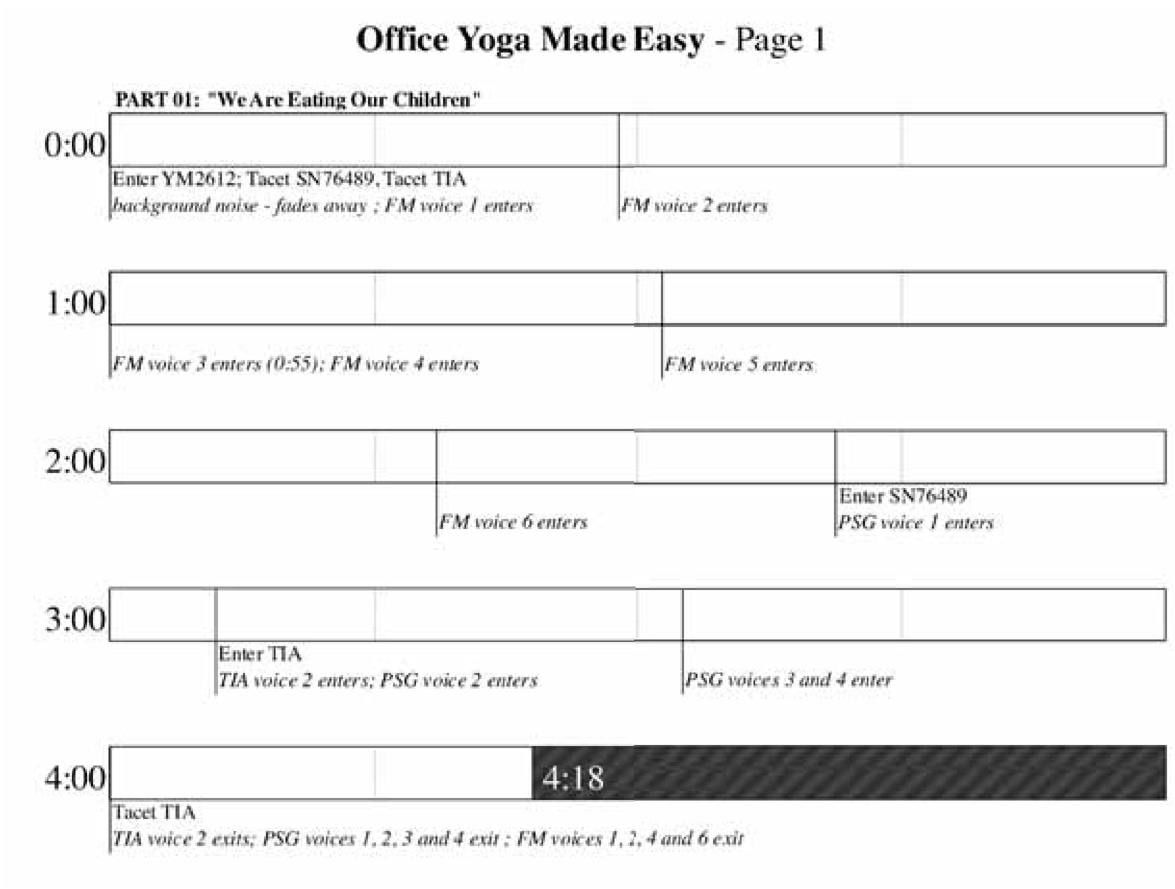
Dynasty - Page 3

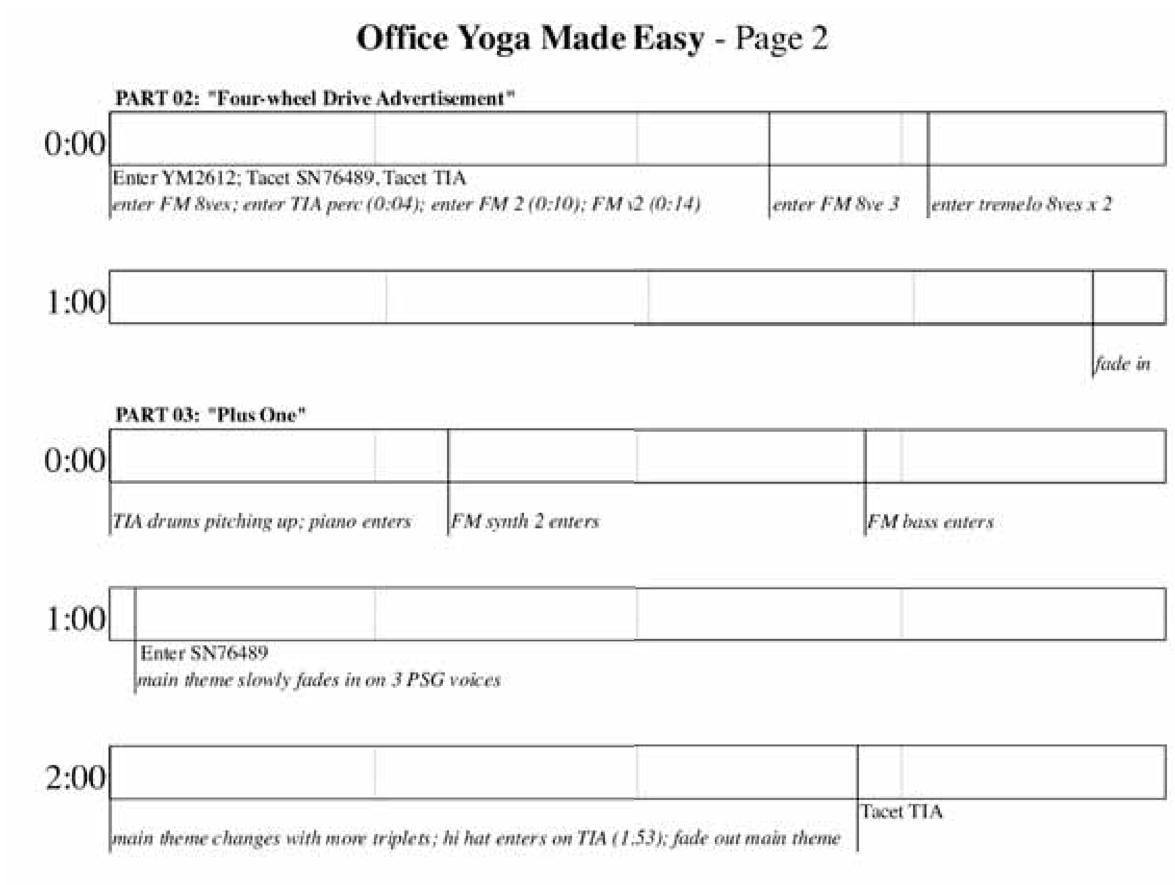

4:00
'verse'; rhythm and bass line change; primary loop
enter secondary melody
5:00
'bridge' triplet feel hi hats
'chorus' with octave tremelo melody
'verse'; additional arpeggios
6:00
enter secondary melody (5:55)
tacet secondary melody (5:55)
'chorus' with octave tremelo melody
'bridge'
7:00
7:25 FINE
TACET 2A03
enter triplet melody line; snare hits (6:55)
organ-like arpeggios

Written Commentary and Timeline: Office Yoga Made Easy (2010)

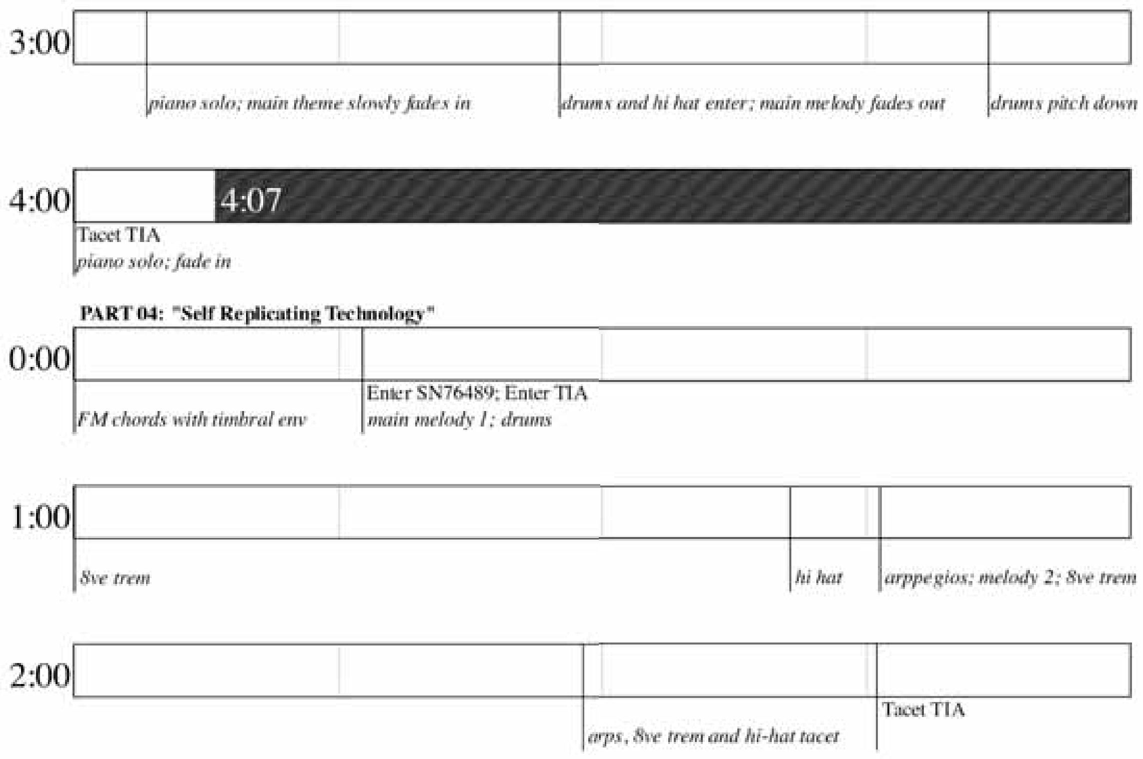
Office Yoga Made Easy incorporates rhythmic and pitched musical phase cycles into the popular compositional idioms, while retaining the minimalist character commonly associated with strict phase-based music. The work is written for the SEGA Nomad Music Interface and the Atari 2600 Interface. SEGA Nomad's backlight generates noise in the audio output. This characteristic is exploited as a positive element by providing a blanket of noise above which the FM synthesis rises. The piano sound (which changes throughout the work) is the product of the simple FM-based synthesis at the core of Nomad. A melody line in the middle of the work makes use of software-based additive synthesis for the Nomad PSG. The result is an introspective work that is meditative, while retaining listener interest through the use of slow-moving melody lines and recognisable percussion phrases.

The timeline for this work is notated using a simplified graphical notation. Every horizontal row represents one minute of audio. Entry and exit points for instruments are shown, as well as structural divisions and musical highlights are marked on the timeline using text.

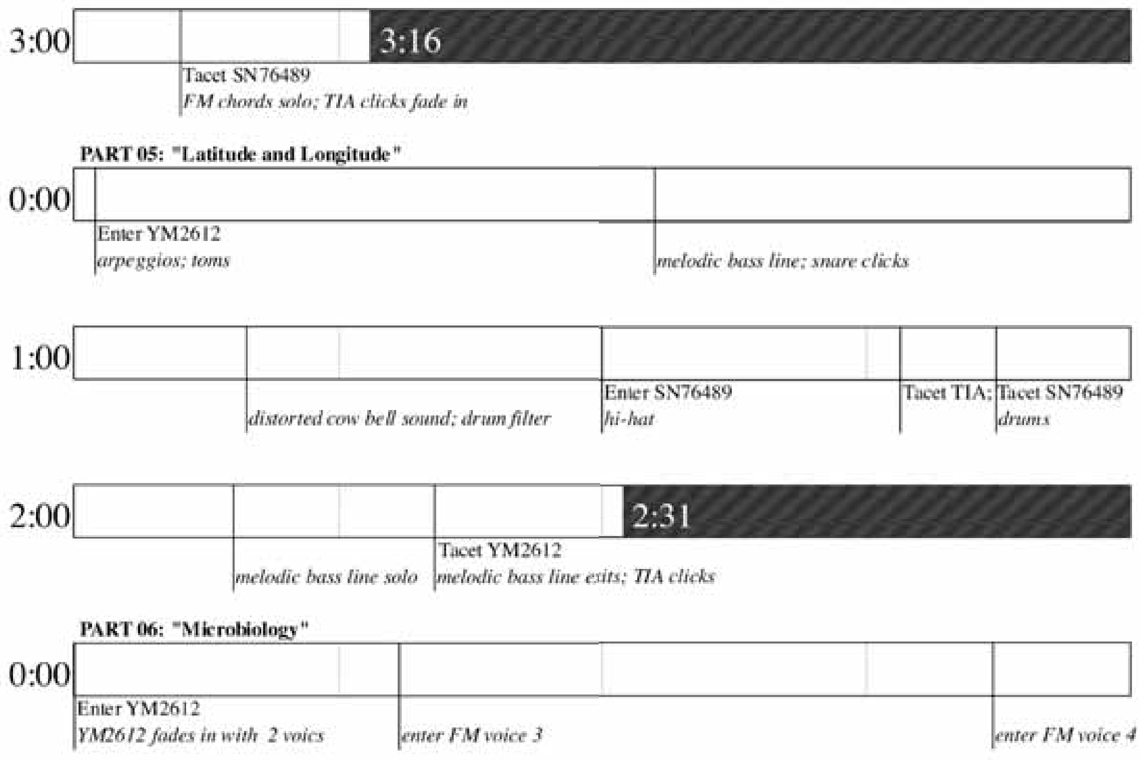

Office Yoga Made Easy - Page 1
PART 01: "We Are Eating Our Children"
0:00
Enter YM2612; Tacet SN76489, Tacet TIA
background noise - fades away ; FM voice 1 enters
FM voice 2 enters
1:00
FM voice 3 enters (0:55); FM voice 4 enters
FM voice 5 enters
2:00
FM voice 6 enters
Enter SN76489
PSG voice 1 enters
3:00
Enter TIA
TIA voice 2 enters; PSG voice 2 enters
PSG voices 3 and 4 enter
4:00
4:18
Tacet TIA
TIA voice 2 exits; PSG voices 1, 2, 3 and 4 exit ; FM voices 1, 2, 4 and 6 exit


Office Yoga Made Easy - Page 2
PART 02: "Four-wheel Drive Advertisement"
0:00
Enter YM2612; Tacet SN76489, Tacet TIA
enter FM 8ves; enter TIA perc (0:04); enter FM 2 (0:10); FM v2 (0:14)
enter FM 8ve 3
enter tremelo 8ves x 2
1:00
fade in
PART 03: "Plus One"
0:00
TIA drums pitching up; piano enters
FM synth 2 enters
FM bass enters
1:00
Enter SN76489
main theme slowly fades in on 3 PSG voices
2:00
main theme changes with more triplets; hi hat enters on TIA (1:53); fade out main theme
Tacet TIA

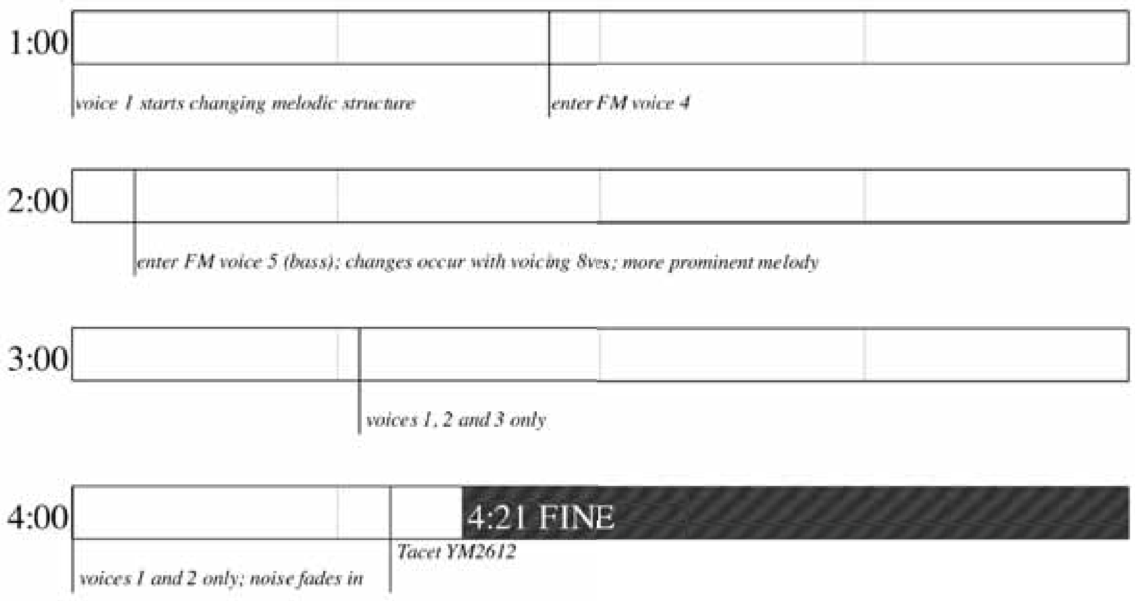
Office Yoga Made Easy - Page 3


3:00
piano solo; main theme slowly fades in
drums and hi hat enter; main melody fades out
drums pitch down
4:00
4:07
Tacet TIA
piano solo; fade in
PART 04: "Self Replicating Technology"
0:00
FM chords with timbral env
Enter SN76489; Enter TIA
main melody 1; drums
1:00
8ve trem
hi hat
arppegios; melody 2; 8ve trem
2:00
arps, 8ve trem and hi-hat tacet
Tacet TIA

Office Yoga Made Easy - Page 4



Office Yoga Made Easy - Page 5

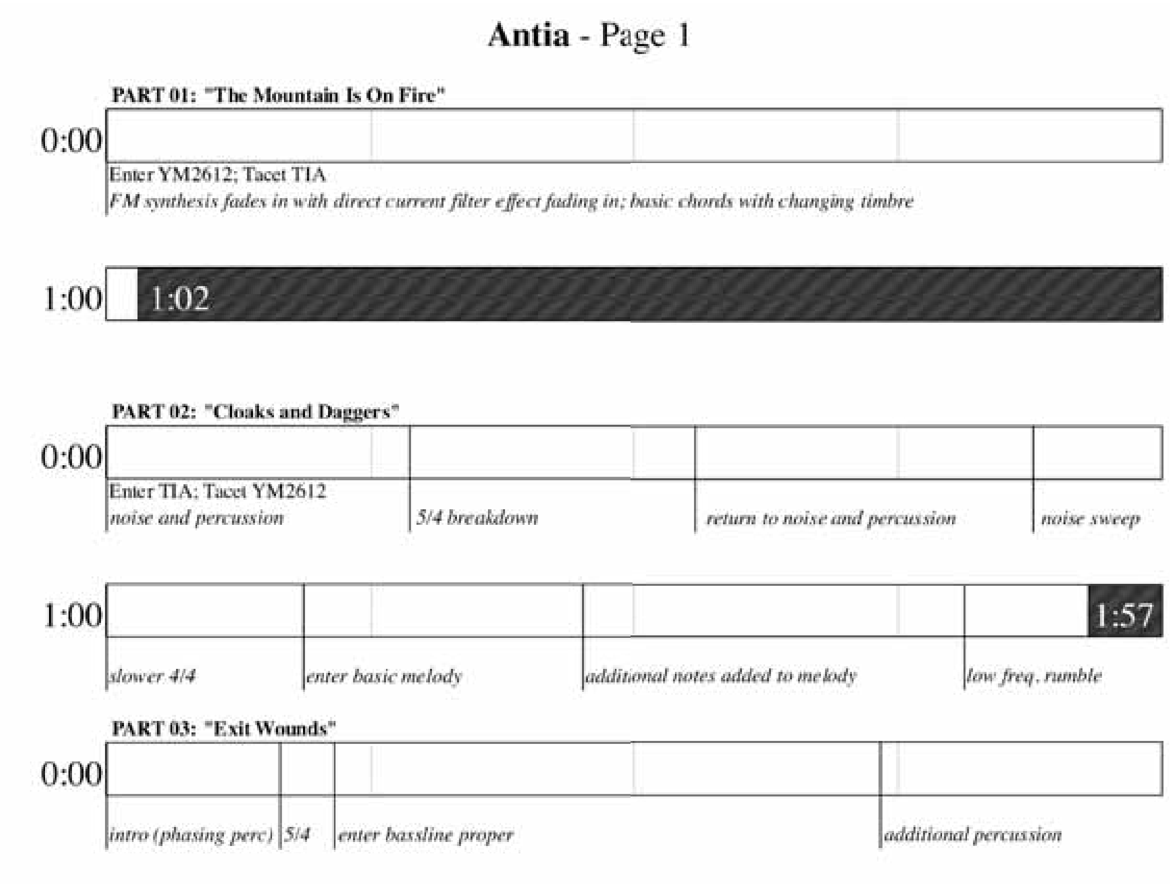

1:00
voice 1 starts changing melodic structure
enter FM voice 4
2:00
enter FM voice 5 (bass); changes occur with voicing 8ves; more prominent melody
3:00
voices 1, 2 and 3 only
4:00
4:21 FINE
Tacet YM2612
voices 1 and 2 only; noise fades in

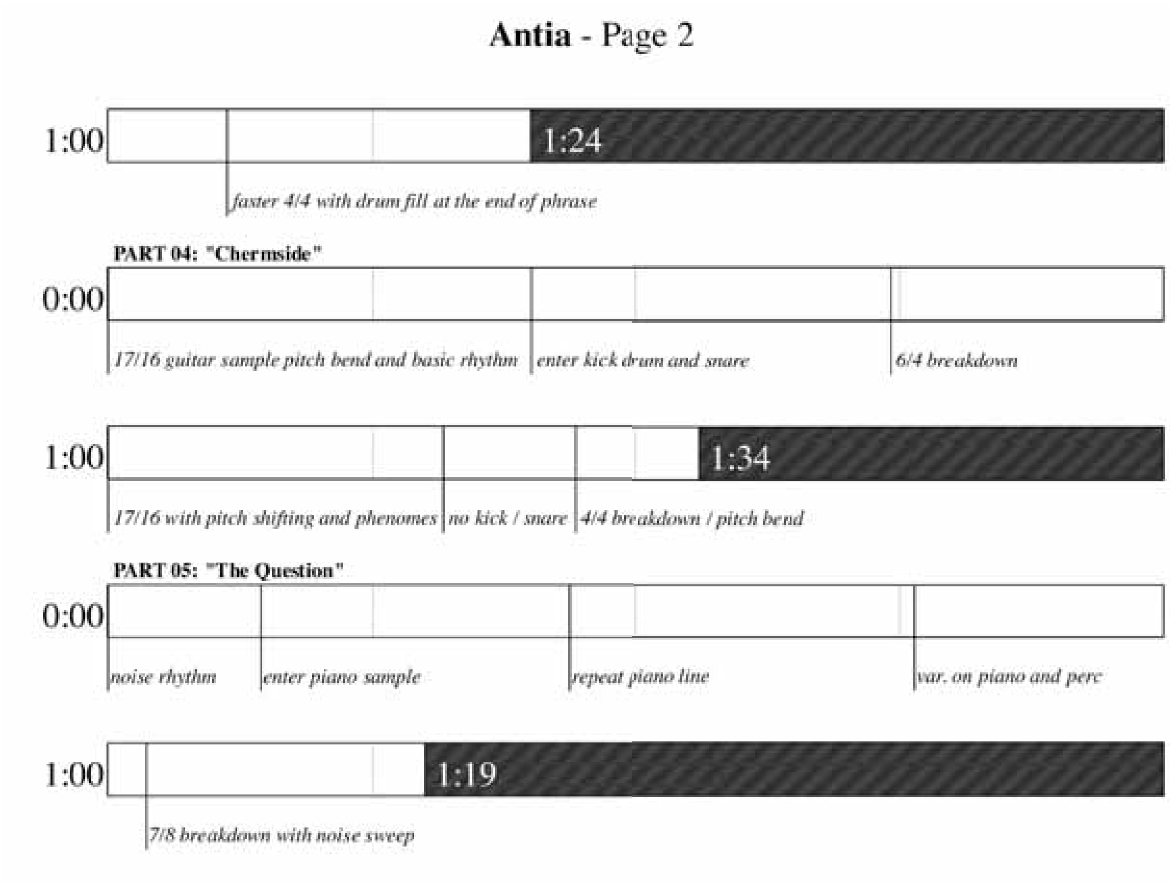
Written Commentary and Timeline: Antia (2009 - 2010)

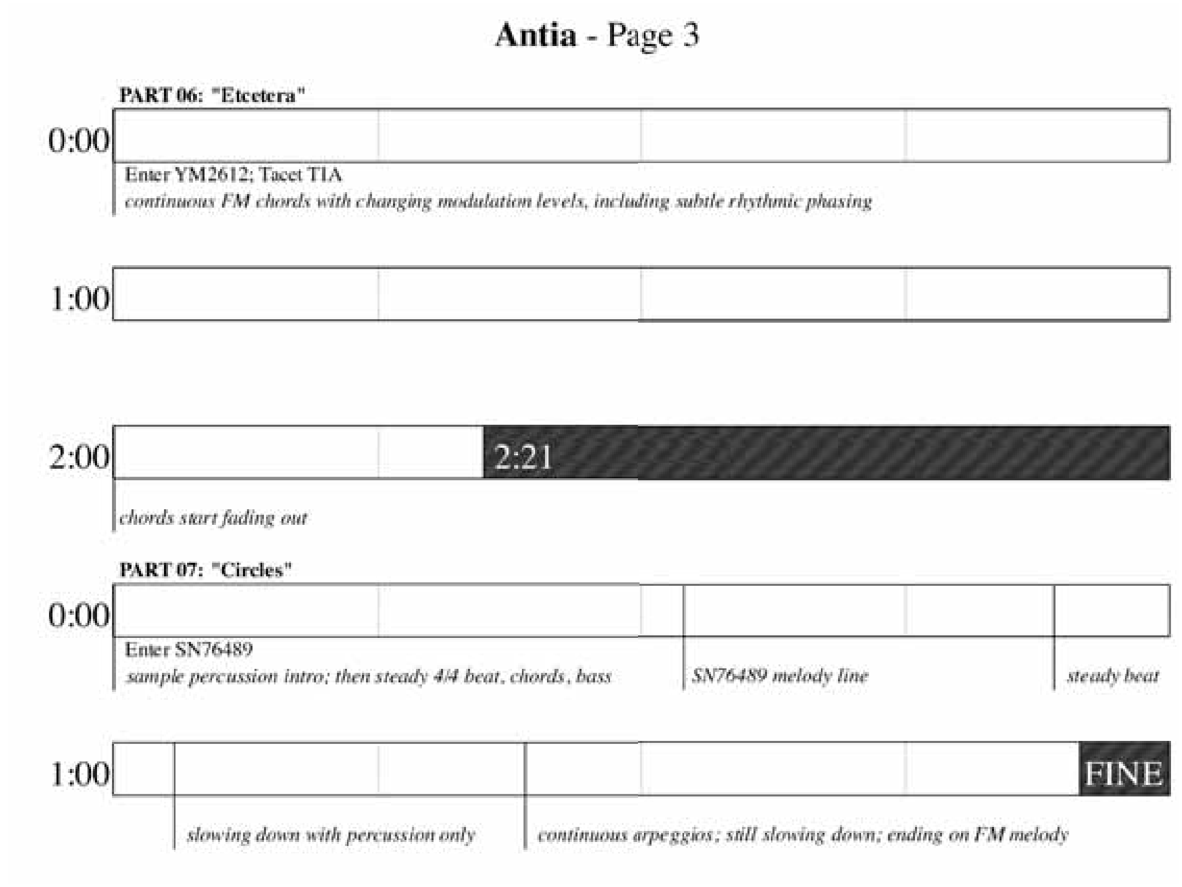
Antia showcases the capacity of the Interface Framework to exploit the contrast between the harshness of the Atari 2600 TIA chip and the soft, gentle timbres possible with the SEGA Mega Drive's YM2612 FM sound chip. The work is made of seven parts, and juxtaposes complex rhythms and fast tempi with smooth textures.

Notable technical features made possible by the Framework include the use of the YM2612 DAC as a filter mechanism, and extensive sample manipulation via the TIA. Timbral shifts play an important role in this work, as the dynamic and interplay from section to section highlights the use of each console. The timeline for this work is notated using a simplified graphical notation. Every horizontal row represents one minute of audio. Entry and exit points for instruments are shown, as well as structural divisions and musical highlights are marked on the timeline using text.

Excerpts from *Antia* have been played live at the 2009 Blip Festival in New York, the 2010 Blip Festival in Tokyo and at the 2010 Sound Bytes 6: Super Stars concert in Melbourne. Additionally, the work has received airplay on local radio stations such as 4Z FM in Brisbane


Antia - Page 1
PART 01: "The Mountain Is On Fire"
0:00
Enter YM2612; Tacet TIA
FM synthesis fades in with direct current filter effect fading in; basic chords with changing timbre
1:00
1:02
PART 02: "Cloaks and Daggers"
0:00
Enter TIA; Tacet YM2612
noise and percussion
5/4 breakdown
return to noise and percussion
noise sweep
1:00
1:57
slower 4/4
enter basic melody
additional notes added to melody
low freq. rumble
PART 03: "Exit Wounds"
0:00
intro (phasing perc)
5/4
enter bassline proper
additional percussion


Antia - Page 2
1:00
1:24
faster 4/4 with drum fill at the end of phrase
PART 04: "Chermside"
0:00
17/16 guitar sample pitch bend and basic rhythm
enter kick drum and snare
6/4 breakdown
1:00
1:34
17/16 with pitch shifting and phenomes
no kick / snare
4/4 breakdown / pitch bend
PART 05: "The Question"
0:00
noise rhythm
enter piano sample
repeat piano line
var. on piano and perc
1:00
1:19
7/8 breakdown with noise sweep


Antia - Page 3
PART 06: "Etcetera"
0:00
Enter YM2612; Tacet TIA
continuous FM chords with changing modulation levels, including subtle rhythmic phasing
1:00
2:00
2:21
chords start fading out
PART 07: "Circles"
0:00
Enter SN76489
sample percussion intro; then steady 4/4 beat, chords, bass
SN76489 melody line
steady beat
1:00
FINE
slowing down with percussion only
continuous arpeggios; still slowing down; ending on FM melody

Written Commentary and Timeline: Error Repeat (2009)

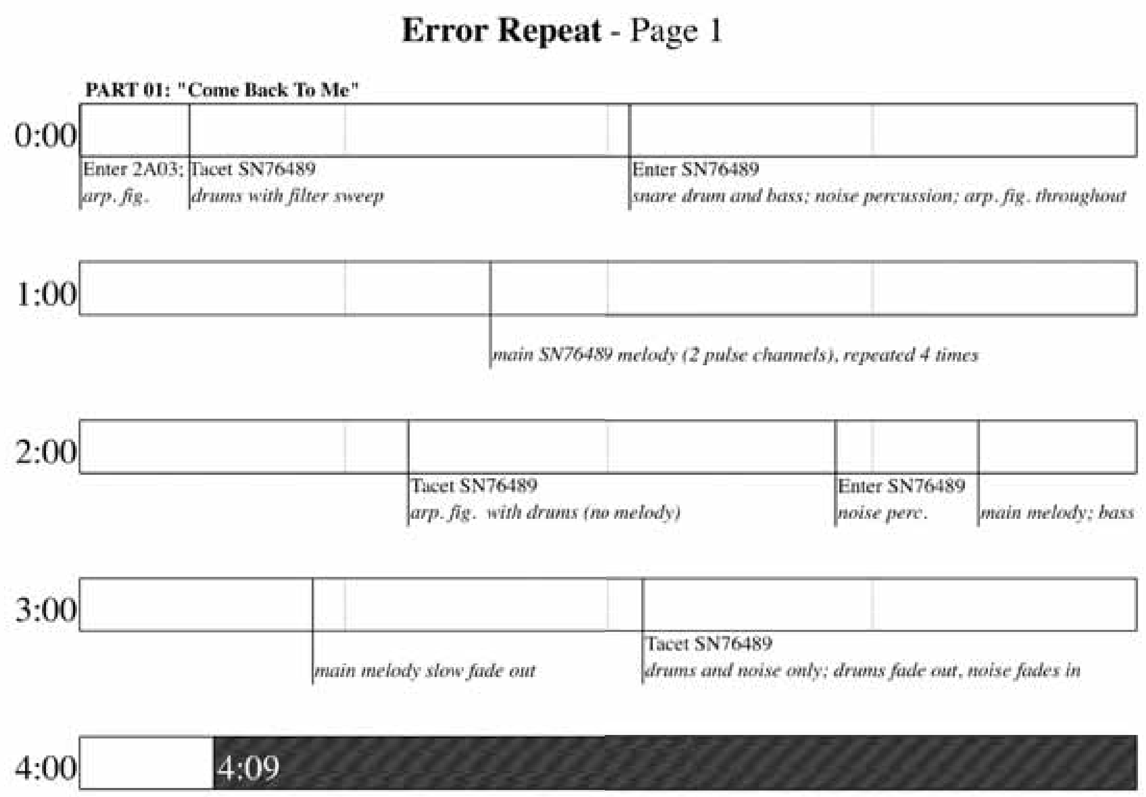
Error Repeat blurs the line between chipmusic as genre and chipmusic as process, and is intended as an homage to exceptional chipmusic composers, such as Tobias Nordloef, Chris Kann (x|k), Paul Slocum (of Tree Wave) and Timothy Lamb (Trash80).

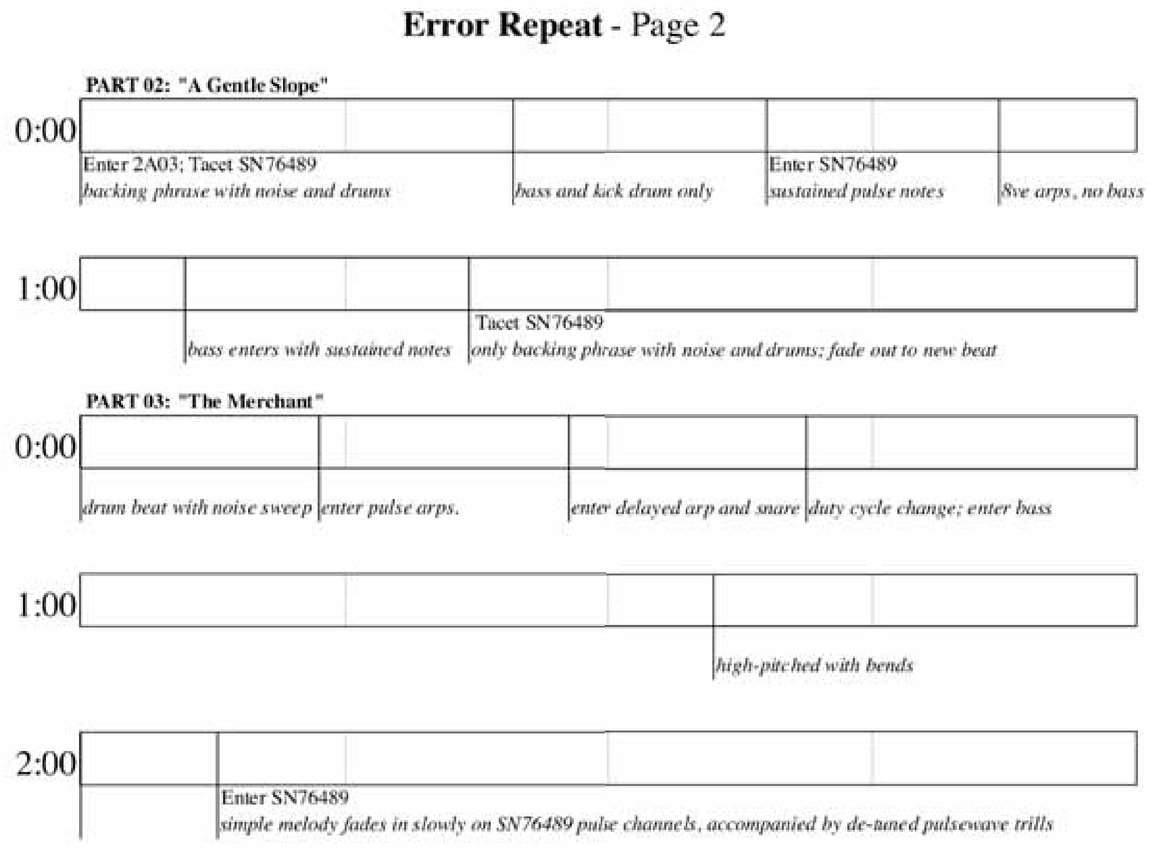
Error Repeat is written for the SEGA Master System (SN76489 PSG) and the Nintendo Entertainment System (2A03), systems that have been merged through the application of the Interface Framework described in the section titled SEGA Master System Music Interface (pp. 47-58). The work successfully fuses the sound of both chips and is a poignant example of combining existing electronic music studio technology with the Interface Framework.

Compositional elements that pay homage to the idea of popular chipmusic include the tempo, which is a steady 145 BPM, as well as the time signature and groove, which is a straight 4/4 with typical kick-snare-kick-snare. The key signatures throughout the work are related to A minor, while the melodies feature syncopated rhythms with punctuating pitch bends layered over straight bass lines. As with the other works featured here, the timeline for this work is notated using a simplified graphical notation. Every horizontal row represents one minute of audio. Entry and exit points for instruments are shown, as well as structural divisions and musical highlights are marked on the timeline using text.

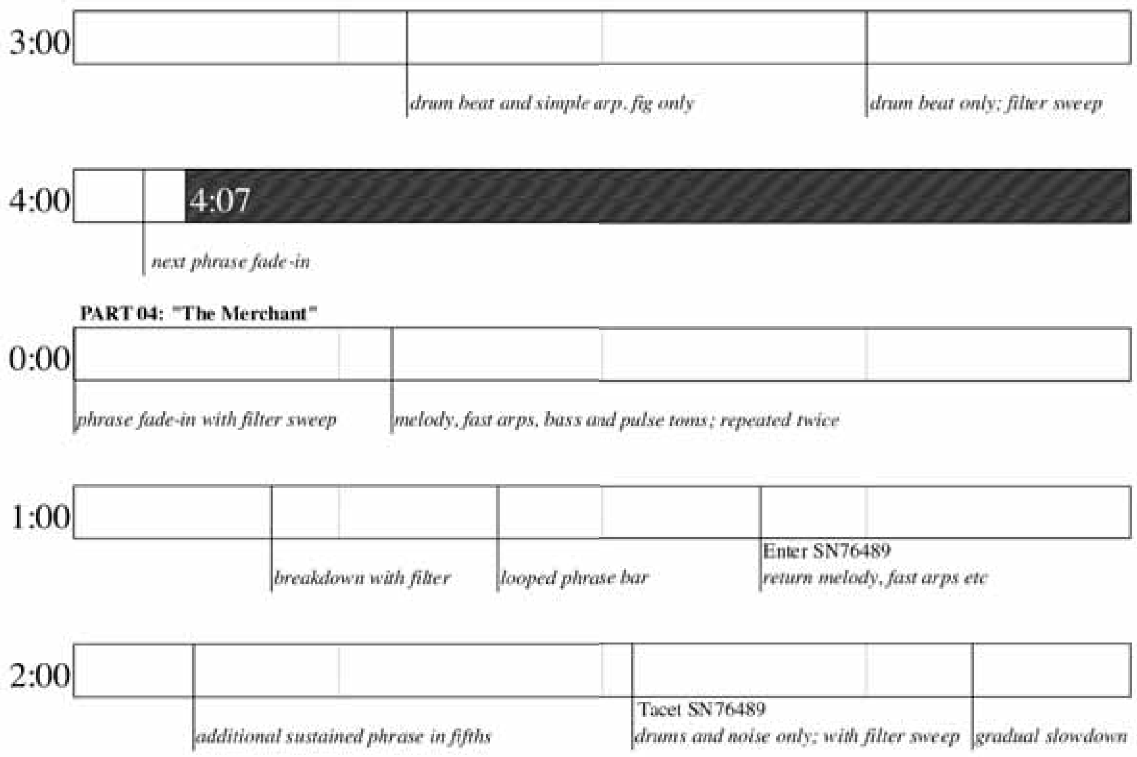
Error Repeat was released on the UK-based chipmusic label Calm Down Kidder in 2009 and was performed at the 2009 Blip Festival in New York City, USA as well as the 2010 Blip Festival in Tokyo, Japan. The work has been received positively, both critically and by listeners. An associate editor of OriginalSoundVersion.com listed *Error Repeat* as the fifth-best chipmusic release of 2009, describing it as:

"...a deeply moving and incredibly cohesive odyssey, told by an unholy union of heavily-modified NES and Sega Master System consoles. Echoing pulsewaves, fleeting arpeggios and steady, thumping beats smoothly switch between forlorn, alienating and adventurous, all the while remaining cool and confident of the road ahead.” (Joshua Kopstein, 18 February 2010).

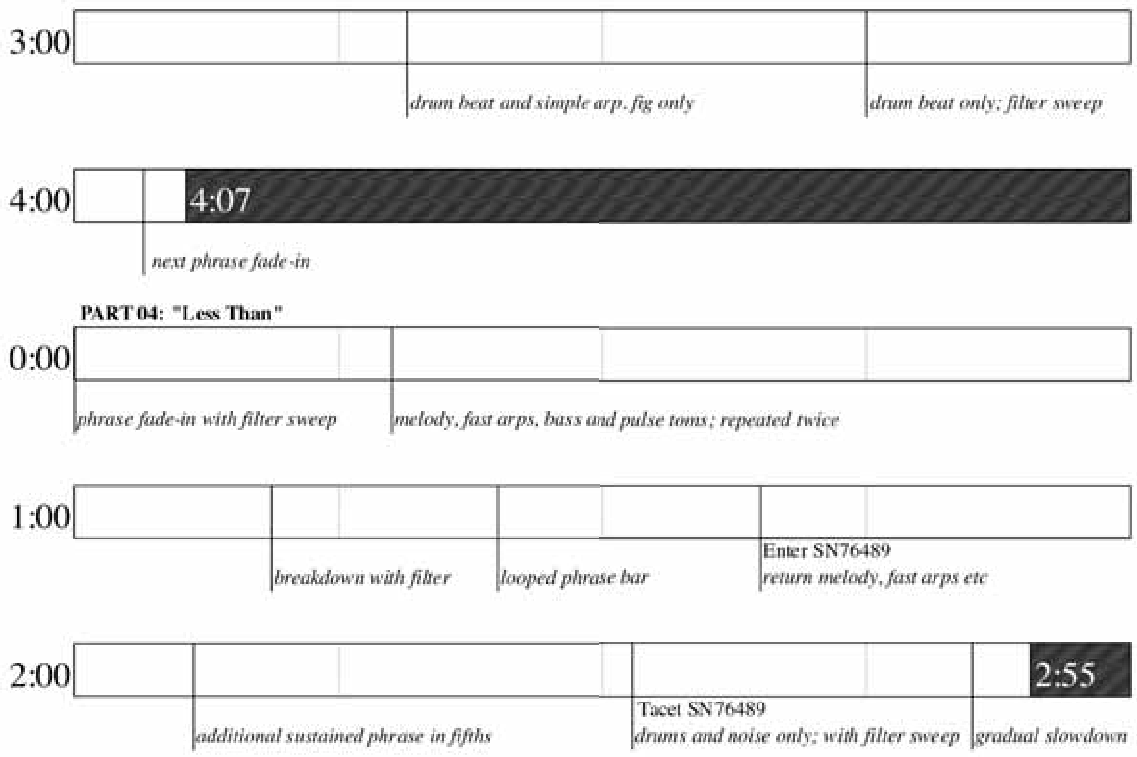

Error Repeat - Page 1
PART 01: "Come Back To Me"
0:00
Enter 2A03;
arp. fig.
Tacet SN76489
drums with filter sweep
Enter SN76489
snare drum and bass; noise percussion; arp. fig. throughout
1:00
main SN76489 melody (2 pulse channels), repeated 4 times
2:00
Tacet SN76489
arp. fig. with drums (no melody)
Enter SN76489
noise perc.
main melody; bass
3:00
main melody slow fade out
Tacet SN76489
drums and noise only; drums fade out, noise fades in
4:00
4:09


Error Repeat - Page 2
PART 02: "A Gentle Slope"
0:00
Enter 2A03; Tacet SN76489
backing phrase with noise and drums
bass and kick drum only
Enter SN76489
sustained pulse notes
8ve arps, no bass
1:00
bass enters with sustained notes
Tacet SN76489
only backing phrase with noise and drums; fade out to new beat
PART 03: "The Merchant"
0:00
drum beat with noise sweep
enter pulse arps.
enter delayed arp and snare
duty cycle change; enter bass
1:00
high-pitched with bends
2:00
Enter SN76489
simple melody fades in slowly on SN76489 pulse channels, accompanied by de-tuned pulsewave trills

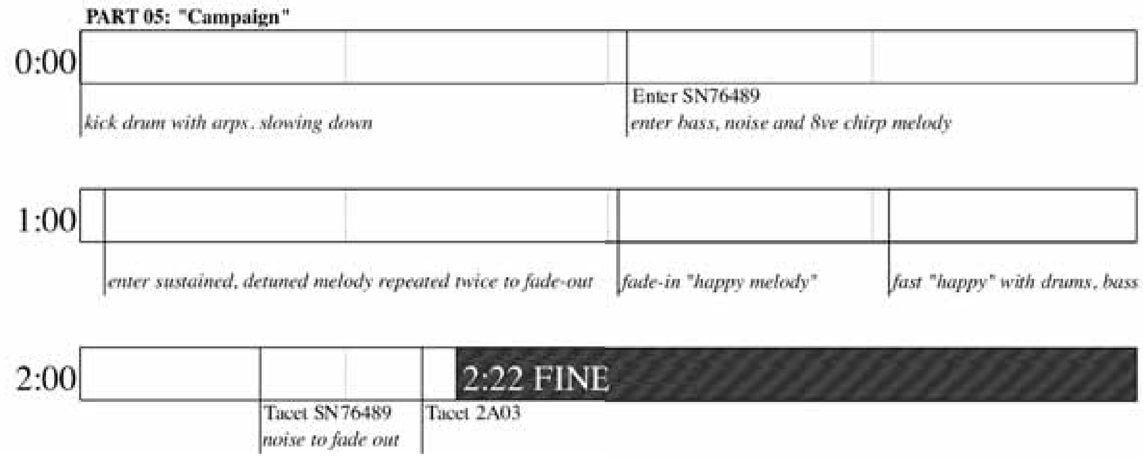
Error Repeat - Page 3


3:00
drum beat and simple arp. fig only
drum beat only; filter sweep
4:00
4:07
next phrase fade-in
PART 04: "The Merchant"
0:00
phrase fade-in with filter sweep
melody, fast arps, bass and pulse toms; repeated twice
1:00
breakdown with filter
looped phrase bar
Enter SN76489
return melody, fast arps etc
2:00
additional sustained phrase in fifths
Tacet SN76489
drums and noise only; with filter sweep
gradual slowdown

Error Repeat - Page 4


3:00
drum beat and simple arp. fig only
drum beat only; filter sweep
4:00
4:07
next phrase fade-in
PART 04: "Less Than"
0:00
phrase fade-in with filter sweep
melody, fast arps, bass and pulse toms; repeated twice
1:00
breakdown with filter
looped phrase bar
Enter SN76489
return melody, fast arps etc
2:00
2:55
additional sustained phrase in fifths
Tacet SN76489
drums and noise only; with filter sweep
gradual slowdown

Error Repeat - Page 5


PART 05: "Campaign"
0:00
Enter SN76489
kick drum with arps. slowing down
enter bass, noise and 8ve chirp melody
1:00
enter sustained, detuned melody repeated twice to fade-out
fade-in "happy melody"
fast "happy" with drums, bass
2:00
2:22 FINE
Tacet SN76489
noise to fade out
Tacet 2A03

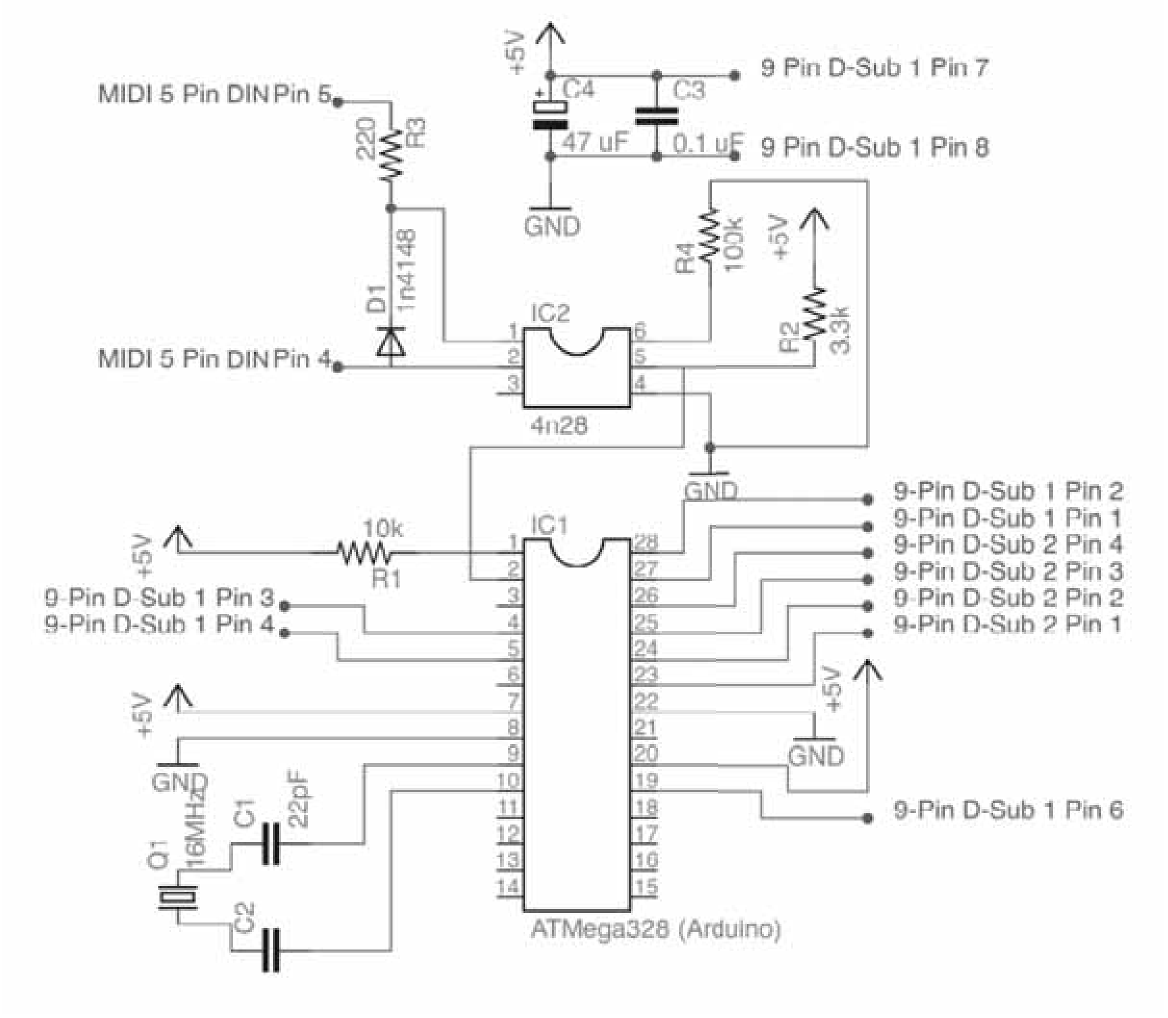
Appendix C: Circuit Schematics

Schematic: Atari 2600 Music Interface

Overview

The Atari 2600 Music Interface is made up of two main parts - the interface hardware (which includes a pre-programmed microcontroller) and a custom program for the Atari 2600 console (which can be put onto a homemade cart or a third-party flash cart). This section covers the schematic and related information for the hardware portion of the interface.

Schematic



Notes

9 Pin D-subminiature 1 and 2 refer to controller ports one and two of the Atari 2600 video game console. IC1 is the ATmega328 microcontroller loaded with an Arduino bootloader and Atari 2600 Music Interface firmware. For reference - when considering construction using a pre-built Arduino board:

- IC1 Pin 2 = Arduino Digital Pin 0 = PORTD Bit 0
- IC1 Pin 3 = Arduino Digital Pin 1 = PORTD Bit 1
- IC1 Pin 4 = Arduino Digital Pin 2 = PORTD Bit 2
- IC1 Pin 5 = Arduino Digital Pin 3 = PORTD Bit 3
- IC1 Pin 6 = Arduino Digital Pin 4 = PORTD Bit 4

- IC1 Pin 11 = Arduino Digital Pin 5 = PORTD Bit 5
- IC1 Pin 12 = Arduino Digital Pin 6 = PORTD Bit 6
- IC1 Pin 13 = Arduino Digital Pin 7 = PORTD Bit 7

- IC1 Pin 14 = Arduino Digital Pin 8 = PORTB Bit 0
- IC1 Pin 15 = Arduino Digital Pin 9 = PORTB Bit 1
- IC1 Pin 16 = Arduino Digital Pin 10 = PORTB Bit 2
- IC1 Pin 17 = Arduino Digital Pin 11 = PORTB Bit 3
- IC1 Pin 18 = Arduino Digital Pin 12 = PORTB Bit 4
- IC1 Pin 19 = Arduino Digital Pin 13 = PORTB Bit 5

- IC1 Pin 23 = Arduino analogue Pin 0 = PORTC Bit 0
- IC1 Pin 24 = Arduino analogue Pin 0 = PORTC Bit 0
- IC1 Pin 25 = Arduino analogue Pin 0 = PORTC Bit 0
- IC1 Pin 26 = Arduino analogue Pin 0 = PORTC Bit 0
- IC1 Pin 27 = Arduino analogue Pin 0 = PORTC Bit 0
- IC1 Pin 28 = Arduino analogue Pin 0 = PORTC Bit 0

Minimum Parts List

ICs:

- ATmega328 pre-programmed with an Arduino bootloader and the Vectrex Music Interface firmware
- 4n28 optocoupler

Resistors:

- 220 Ohms
- 3.3k Ohms
- 10k Ohms
- 100k Ohms

Capacitors:

- 22 pF
- 22 pF
- 0.1 uF
- 47 uF

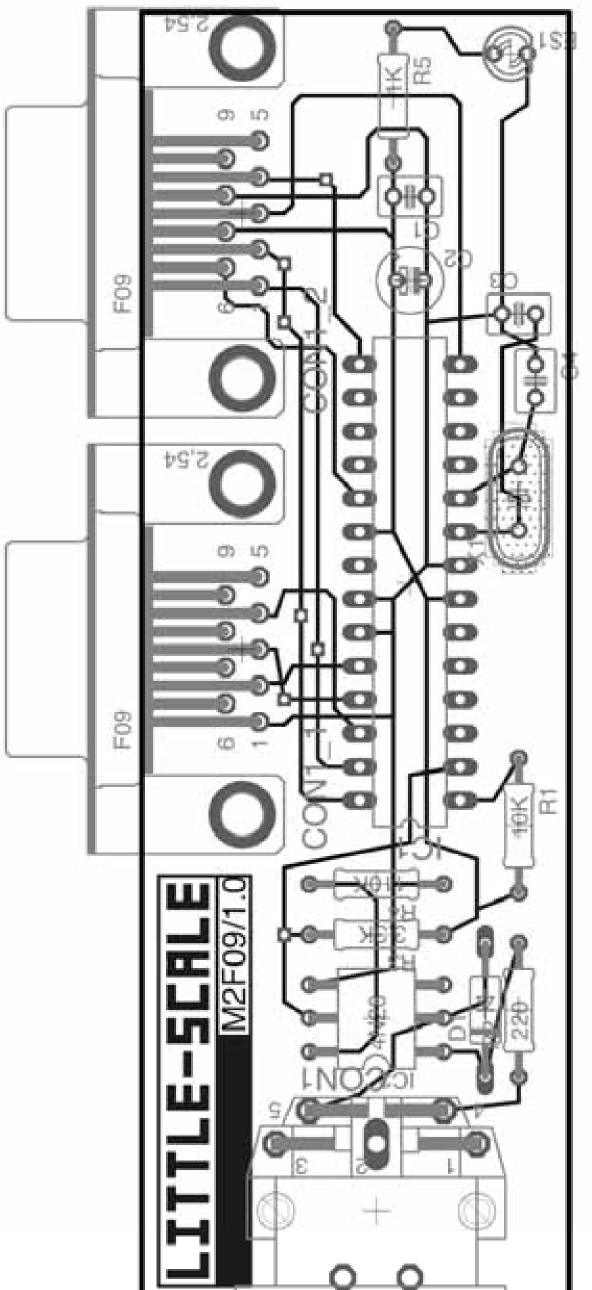
Miscellaneous:

- 16.00 MHz crystal
- 1n4148 diode

Connectors:

- 5 pin DIN
- 9 pin D-Sub
- 9 pin D-Sub

PCB Example

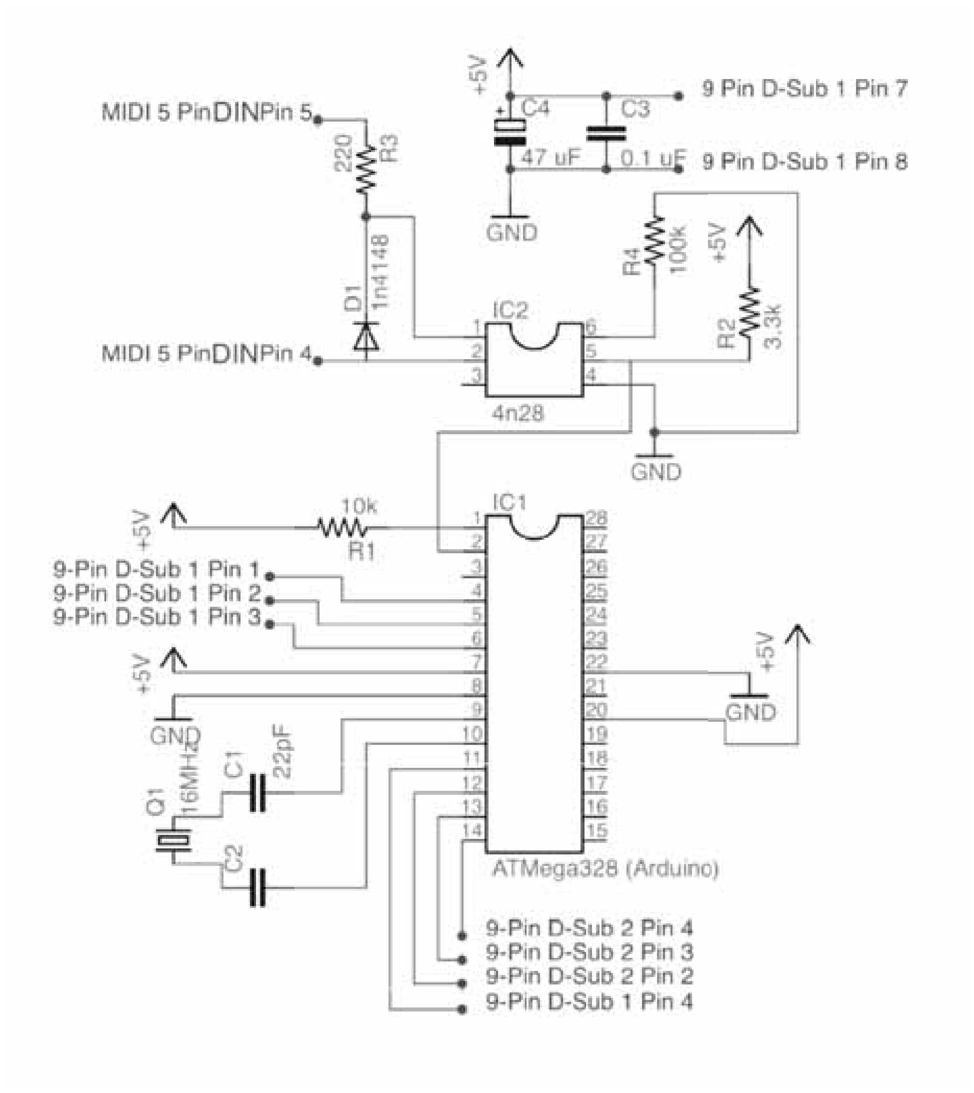

LITTLE-SCALE
M2F09/1.0
F09
F09
CON1_1
CON1_2
IC1
IC2
ICCON1
R1
10K
R5
1K
C1
C2
C3
C4
X1
ES1
D1
220
6N139
2.54
2.54

Schematic: Vectrex Music Interface

Overview

The Vectrex Music Interface is made up of two main parts - the interface hardware (which includes a pre-programmed microcontroller) and a custom program for the Vectrex console (which can be put onto a homemade cart or a third-party flash cart). This section covers the schematic and related information for the hardware portion of the interface.

Schematic



Notes

9 Pin D-subminiature 1 and 2 refer to controller ports one and two of the Vectrex video game console. IC1 is the ATmega328 microcontroller loaded with an Arduino bootloader

and Vectrex Music Interface firmware. For reference - when considering construction using a pre-built Arduino board:

- IC1 Pin 2 = Arduino Digital Pin 0 = PORTD Bit 0
- IC1 Pin 3 = Arduino Digital Pin 1 = PORTD Bit 1
- IC1 Pin 4 = Arduino Digital Pin 2 = PORTD Bit 2
- IC1 Pin 5 = Arduino Digital Pin 3 = PORTD Bit 3
- IC1 Pin 6 = Arduino Digital Pin 4 = PORTD Bit 4
- IC1 Pin 11 = Arduino Digital Pin 5 = PORTD Bit 5
- IC1 Pin 12 = Arduino Digital Pin 6 = PORTD Bit 6
- IC1 Pin 13 = Arduino Digital Pin 7 = PORTD Bit 7

- IC1 Pin 14 = Arduino Digital Pin 8 = PORTB Bit 0
- IC1 Pin 15 = Arduino Digital Pin 9 = PORTB Bit 1
- IC1 Pin 16 = Arduino Digital Pin 10 = PORTB Bit 2
- IC1 Pin 17 = Arduino Digital Pin 11 = PORTB Bit 3
- IC1 Pin 18 = Arduino Digital Pin 12 = PORTB Bit 4
- IC1 Pin 19 = Arduino Digital Pin 13 = PORTB Bit 5

- IC1 Pin 23 = Arduino analogue Pin 0 = PORTC Bit 0
- IC1 Pin 24 = Arduino analogue Pin 0 = PORTC Bit 0
- IC1 Pin 25 = Arduino analogue Pin 0 = PORTC Bit 0
- IC1 Pin 26 = Arduino analogue Pin 0 = PORTC Bit 0
- IC1 Pin 27 = Arduino analogue Pin 0 = PORTC Bit 0
- IC1 Pin 28 = Arduino analogue Pin 0 = PORTC Bit 0

Minimum Parts List

ICs:

- ATmega328 pre-programmed with an Arduino bootloader and the Vectrex Music Interface firmware
- 4n28 optocoupler

Resistors:

- 220 Ohms
- 3.3k Ohms
- 10k Ohms
- 100k Ohms

Capacitors:

- 22 pF
- 22 pF
- 0.1 uF
- 47 uF

Miscellaneous:

- 16.00 MHz crystal
- 1n4148 diode

Connectors:

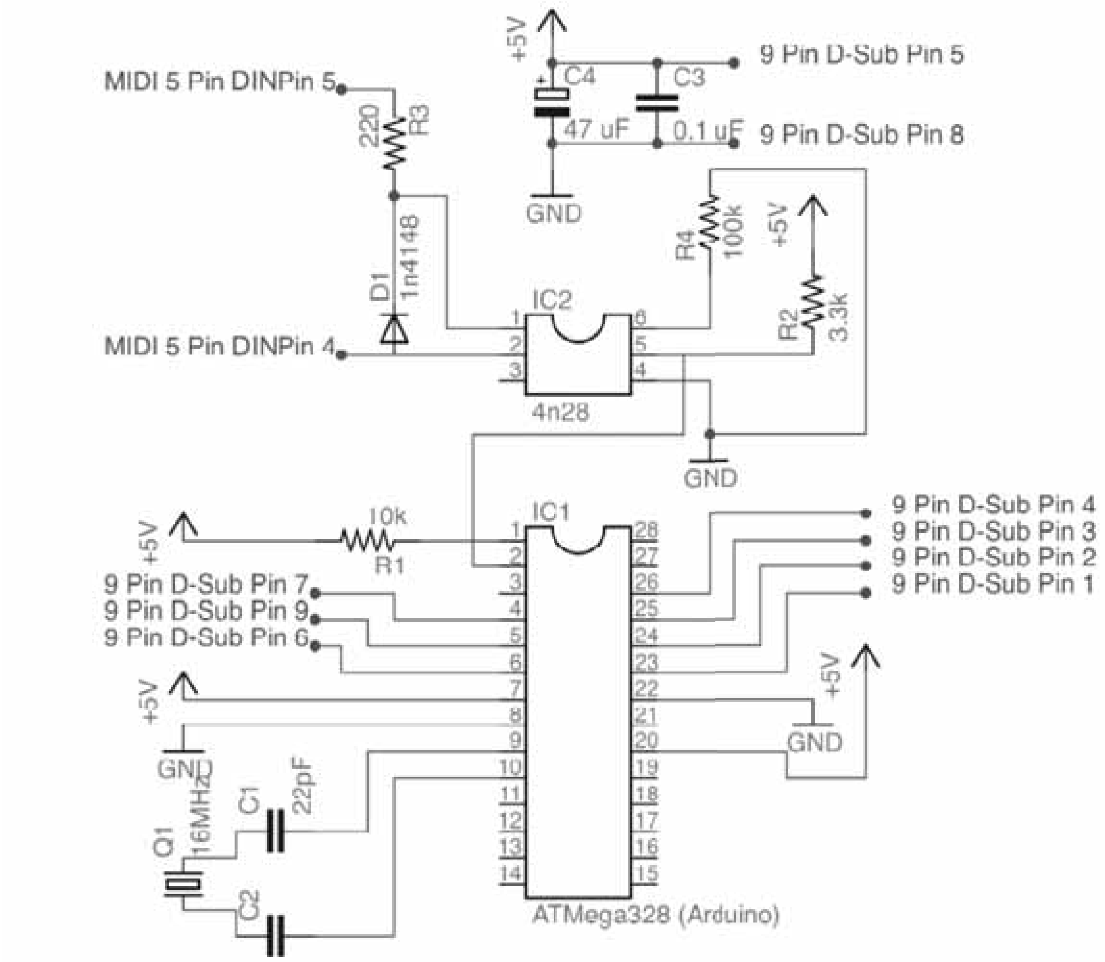
- 5 pin DIN
- 9 pin D-Sub
- 9 pin D-Sub

Schematic: SEGA Master System Music Interface

Overview

The SEGA Master System Music Interface is made up of two main parts - the interface hardware (which includes a pre-programmed microcontroller) and a custom program for the SEGA Master System console (which can be put onto a homemade cart or a third-party flash cart). This section covers the schematic and related information for the hardware portion of the interface.

Schematic



Notes

9 Pin D-subminiature refers to controller port two of the SEGA Master System video game console. IC1 is the ATmega328 microcontroller loaded with an Arduino bootloader and SEGA Master System Music Interface firmware. For reference - when considering construction using a pre-built Arduino board:

- IC1 Pin 2 = Arduino Digital Pin 0 = PORTD Bit 0
- IC1 Pin 3 = Arduino Digital Pin 1 = PORTD Bit 1
- IC1 Pin 4 = Arduino Digital Pin 2 = PORTD Bit 2

- IC1 Pin 5 = Arduino Digital Pin 3 = PORTD Bit 3
- IC1 Pin 6 = Arduino Digital Pin 4 = PORTD Bit 4
- IC1 Pin 11 = Arduino Digital Pin 5 = PORTD Bit 5
- IC1 Pin 12 = Arduino Digital Pin 6 = PORTD Bit 6
- IC1 Pin 13 = Arduino Digital Pin 7 = PORTD Bit 7

- IC1 Pin 14 = Arduino Digital Pin 8 = PORTB Bit 0
- IC1 Pin 15 = Arduino Digital Pin 9 = PORTB Bit 1
- IC1 Pin 16 = Arduino Digital Pin 10 = PORTB Bit 2
- IC1 Pin 17 = Arduino Digital Pin 11 = PORTB Bit 3
- IC1 Pin 18 = Arduino Digital Pin 12 = PORTB Bit 4
- IC1 Pin 19 = Arduino Digital Pin 13 = PORTB Bit 5

- IC1 Pin 23 = Arduino analogue Pin 0 = PORTC Bit 0
- IC1 Pin 24 = Arduino analogue Pin 0 = PORTC Bit 0
- IC1 Pin 25 = Arduino analogue Pin 0 = PORTC Bit 0
- IC1 Pin 26 = Arduino analogue Pin 0 = PORTC Bit 0
- IC1 Pin 27 = Arduino analogue Pin 0 = PORTC Bit 0
- IC1 Pin 28 = Arduino analogue Pin 0 = PORTC Bit 0

Minimum Parts List

ICs:

- ATmega328 pre-programmed with an Arduino bootloader and the GENMDM firmware

- 4n28 optocoupler

Resistors:

- 220 Ohms
- 3.3k Ohms
- 10k Ohms
- 100k Ohms

Capacitors:

- 22 pF
- 22 pF
- 0.1 uF
- 47 uF

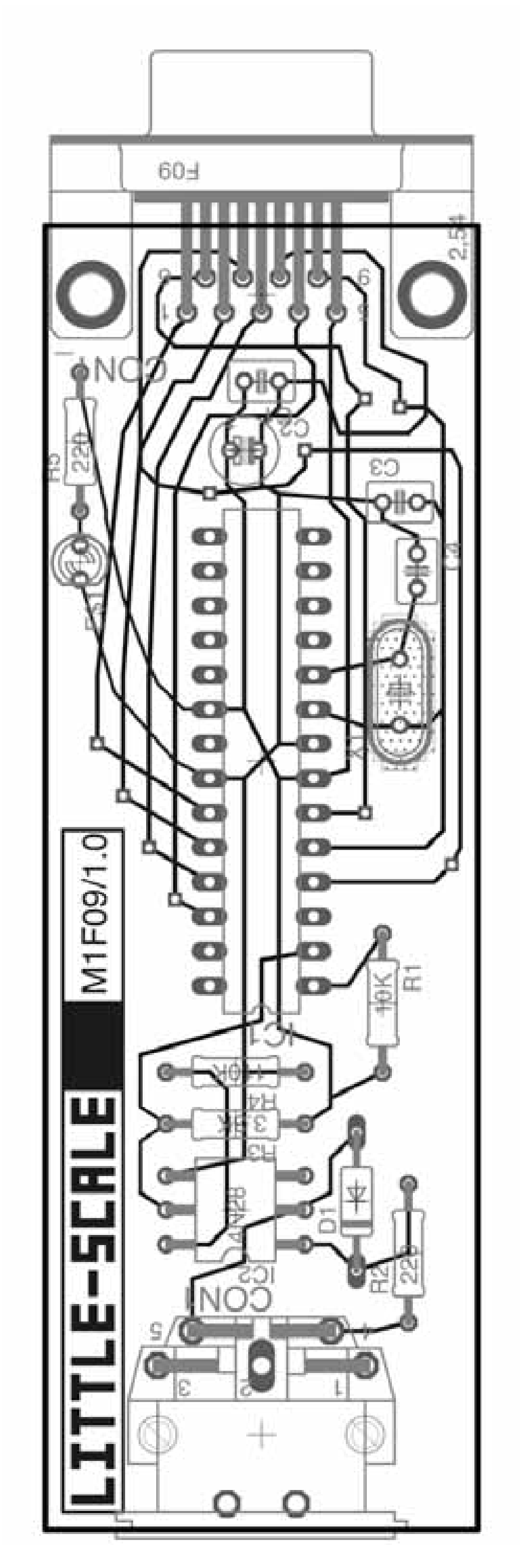
Miscellaneous:

- 16.00 MHz crystal
- 1n4148 diode

Connectors:

- 5 pin DIN
- 9 pin D-Sub

PCB Example

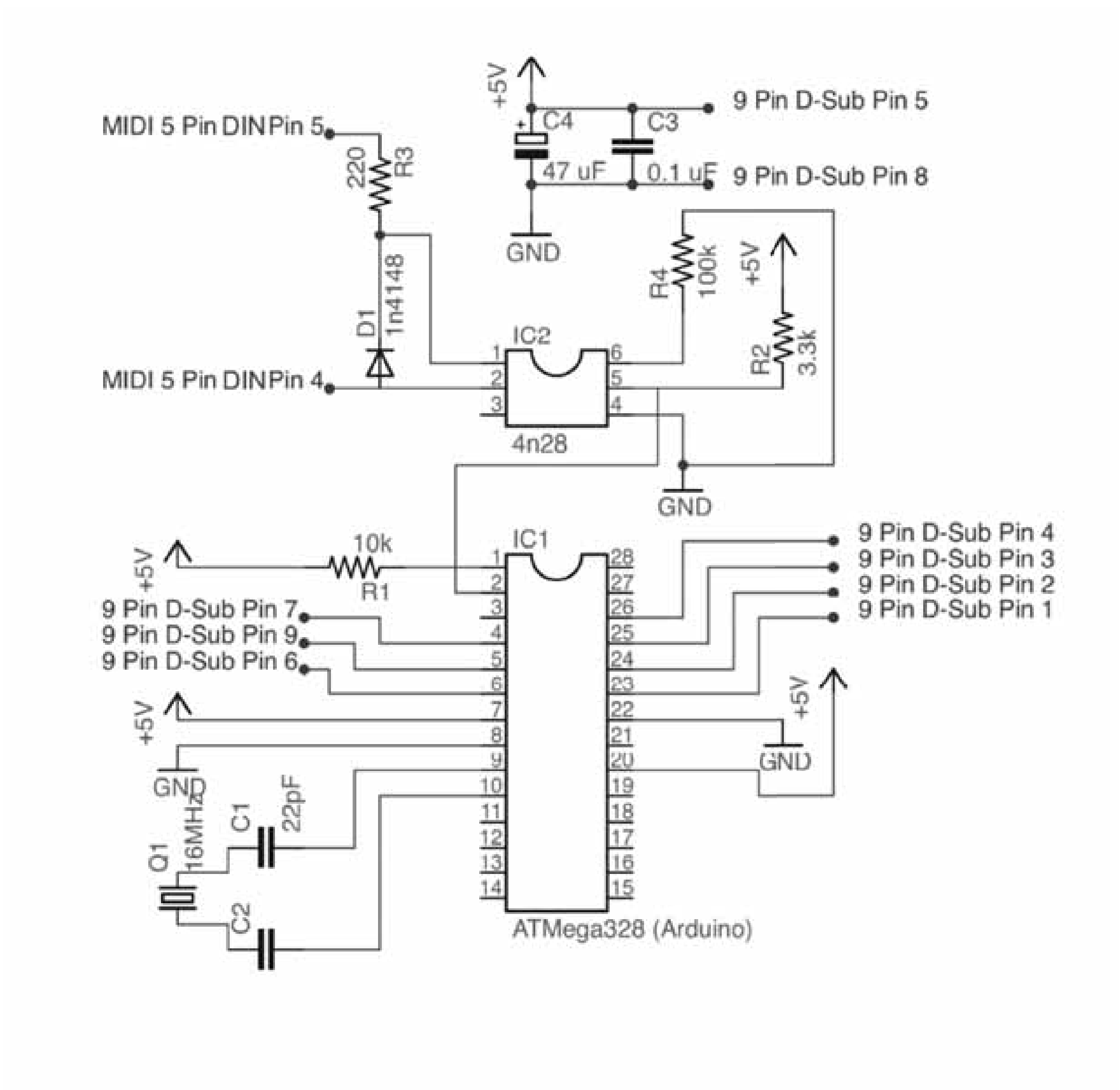


Schematic: SEGA Mega Drive / Genesis Music Interface

Overview

The SEGA Mega Drive / Genesis Music Interface is made up of two main parts - the interface hardware (which includes a pre-programmed microcontroller) and a custom program for the SEGA Mega Drive console (which can be put onto a homemade cart or a third-party flash cart). This section covers the schematic and related information for the hardware portion of the interface.

Schematic



Notes

9 Pin D-subminiature refers to controller port two of the SEGA Mega Drive / Genesis video game console. IC1 is the ATmega328 microcontroller loaded with an Arduino bootloader and the SEGA Mega Drive / Genesis Music Interface firmware. For reference - when considering construction using a pre-built Arduino board:

- IC1 Pin 2 = Arduino Digital Pin 0 = PORTD Bit 0
- IC1 Pin 3 = Arduino Digital Pin 1 = PORTD Bit 1
- IC1 Pin 4 = Arduino Digital Pin 2 = PORTD Bit 2
- IC1 Pin 5 = Arduino Digital Pin 3 = PORTD Bit 3
- IC1 Pin 6 = Arduino Digital Pin 4 = PORTD Bit 4
- IC1 Pin 11 = Arduino Digital Pin 5 = PORTD Bit 5
- IC1 Pin 12 = Arduino Digital Pin 6 = PORTD Bit 6
- IC1 Pin 13 = Arduino Digital Pin 7 = PORTD Bit 7

- IC1 Pin 14 = Arduino Digital Pin 8 = PORTB Bit 0
- IC1 Pin 15 = Arduino Digital Pin 9 = PORTB Bit 1
- IC1 Pin 16 = Arduino Digital Pin 10 = PORTB Bit 2
- IC1 Pin 17 = Arduino Digital Pin 11 = PORTB Bit 3
- IC1 Pin 18 = Arduino Digital Pin 12 = PORTB Bit 4
- IC1 Pin 19 = Arduino Digital Pin 13 = PORTB Bit 5

- IC1 Pin 23 = Arduino analogue Pin 0 = PORTC Bit 0
- IC1 Pin 24 = Arduino analogue Pin 0 = PORTC Bit 0
- IC1 Pin 25 = Arduino analogue Pin 0 = PORTC Bit 0

- IC1 Pin 26 = Arduino analogue Pin 0 = PORTC Bit 0
- IC1 Pin 27 = Arduino analogue Pin 0 = PORTC Bit 0
- IC1 Pin 28 = Arduino analogue Pin 0 = PORTC Bit 0

Minimum Parts List

ICs:

- ATmega328 pre-programmed with an Arduino bootloader and the SEGA Mega Drive / Genesis Music Interface firmware
- 4n28 optocoupler

Resistors:

- 220 Ohms
- 3.3k Ohms
- 10k Ohms
- 100k Ohms

Capacitors:

- 22 pF
- 22 pF
- 0.1 uF
- 47 uF

Miscellaneous:

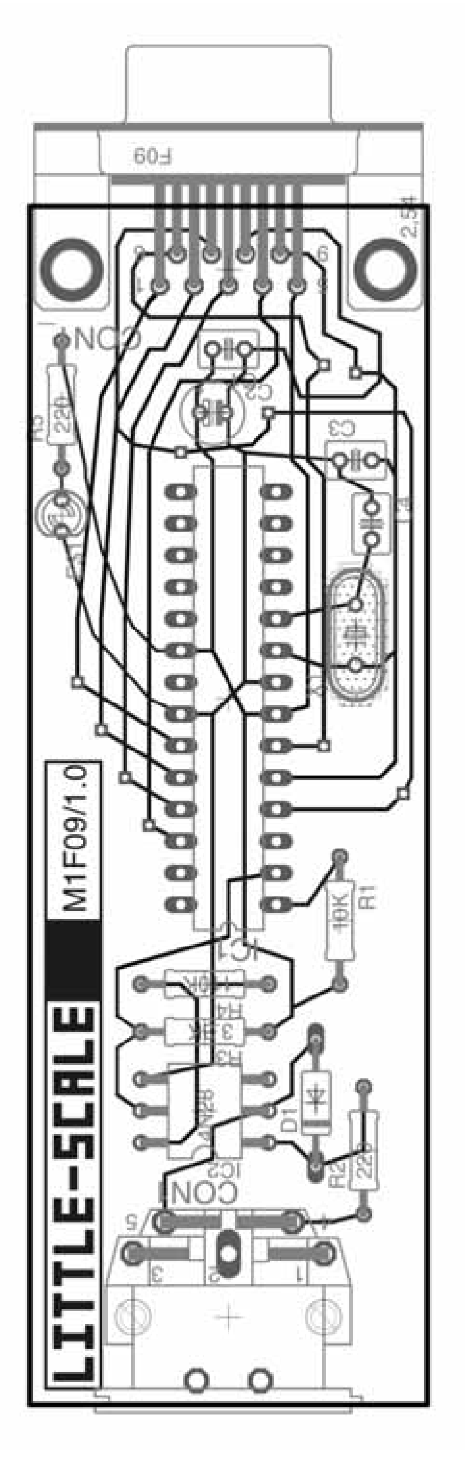
- 16.00 MHz crystal
- 1n4148 diode

Connectors

- 5 pin DIN

◦ 9 pin D-Sub

PCB Example



Appendices D and E are available on separate CD-Roms and are held in the University of Adelaide Library.

Bibliography

‘Adding a Sound Chip’. http://www.onastick.clara.co.uk/sound.htm (accessed 22 March 2010).

‘Arduino: Arduino Board Duemilanove’. http://arduino.cc/en/Main/ArduinoBoardDuemilanove (accessed 16 March 2010).

Arduino. http://arduino.cc/ (accessed 16 March 2010).

Atari Corporation. ‘TIA Schematic. Page 4 of 5’. *http://www.atariage.com/2600/archives/schematics_tia/TIA_1A_400dpi_4.zip* (accessed 15 March 2010).

Carlsson, Anders. ‘Chipmusic’ in *Chipflip*. http://chipflip.wordpress.com/chipmusic/. Accessed 16 March 2010.

Collins, Karen. *Game Sound: An Introduction to the History, Theory, and Practice of Video Game Music and Sound Design.* Cambridge: MIT Press, 2008.

Crawford, Chris; Winner, Lane; Cox, Jim; Chen, Amy; Dunion, Jim; Pitta, Kathleen; Fraser, Bob; and Makrea, Gus. ‘De Re Atari - Chapter 7’, in *Atari Archives: Software and Information*. 1982. http://www.atariarchives.org/dere/chapt07.php (20 March 2010).

Dale, Nell and Lewis, John. ‘Vector Representation of Graphics’, in *Computer Science Illuminated.* 4th ed. Boston: Jones and Bartlett Publishers, 2010.

Don Mahurin. ‘Atari POKEY to Csound Conversion’. December 2006. http://pokey.openright.org/ (accessed 22 March 2010).

Driscoll, Kevin and Diaz, Joshua. ‘Endless Loop: A Brief History of Chiptunes’, in *Transformative Works and Cultures,* vol. 2, 2009. http://journal.transformativeworks.org/index.php/twc/article/view/96/94 (accessed 15 March 2010).

GCE Corporation. *Melody Master*. Software for Vectrex. 1983.

General Instruments Semiconductor Group. ‘AY38910, AY38912, AY38913 Datasheet’. http://www.msxarchive.nl/pub/msx/mirrors/hanso/datasheets/chipsay38910.pdf (accessed 16 March 2010).

General Instruments Semiconductor Group. ‘SP0256-A Datasheet’. http://lib.store.yahoo.net/lib/e-clec-tech/spo256.pdf (accessed 22 March 2010).

Grand, Joe and Russell, Ryan. *Hardware Hacking: Have Fun While Voiding Your Warranty*. St. Louis: Elsevier Inc., 2004.

Grand, Joe; Thornton, Frank and Yarusso, Albert. *Game Console Hacking: Have Fun While Voiding Your Warranty*. St. Louis: Elsevier Inc, 2004.

Huss, Fredrik and Högdahl, Magnus. Fast Tracker 2. Software for DOS.

http://www.gwinternet.com/music/ft2/software/ft2.htm (accessed 10 March 2010).

Kohler, Chris. *Retro Gaming Hacks*. Sebastopol: O'Reilly Publishing, 2005.

Konrad, Martin. *MOD2PSG.* Software for Windows OS. 2007. Published by KonTechs Limited. http://www.kontechs.de/product?name=mod2psg. (accessed 16 March 2010).

Kotlinksi. Johan and Lamb, Timothy. 'Game Boy Prosound Modification'. http://www.littlesounddj.com/lsd/prosound/ (accessed 22 March 2010).

Kotlinski, Johan. *Little Sound DJ.* Software for Game Boy. http://www.littlesounddj.com/lsd/ (accessed 15 March 2010).

Laing, Gordon. *Digital Retro: The Evolution and Design of the Personal Computer.* London: Ilex Publishing, 2004.

Marks, Aaron. 'An Introduction to Game Audio', in *The Complete Guide to Game Audio: For Composers, Musicians, Sound Designers, Game Developers*. 2nd ed. Burlington: Focal Press, 2008.

Molnar, Steven and Fuchs, Henry. 'Advanced Raster Graphics Architecture', in *Computer Graphics: Principles and Practice.* James D. Foley (ed). 2nd ed. Reading: Addison-Wesley Professional, 1997.

Montfort, Nick and Bogost, Ian. *Racing the Beam: The Atari Video Computer System*. Cambridge: MIT Press, 2009.

Parker, Phillip. *Calculators: Webster's Quotations, Facts and Phrases.* San Diego: Icon Group International, 2008.

Parker, Phillip. *Chips: Webster's Quotations, Facts and Phrases.* San Diego: Icon Group International, 2008.

Paul Théberge. *Any Sound You Can Imagine: Making Music / Consuming Technology.* Middletown: Wesleyan University Press, 1997.

Phillips Corporation. Phillips SAA-1099 Complex Programmable Sound Generator Datasheet. http://velesoft.speccy.cz/samcoupe/saa1099/saa-1099_data_sheet.zip (accessed 22 March 2010).

Reunanen, Markuu and Silvast, Antti. 'Demoscene Platforms: A Case Study on the Adoption of Home Computers', in *History of Nordic Computing 2: Second IFIP WG 9.7 Conference, HiNC 2*. Turku, Finland. August 21-23, 2007. New York: Springer Publishing, 2009. 289 – 301.

Salomon, Chris 'An Introduction to Vectrex Programming: Appendix A: Overview of Functional Classes: 12: Sound'. http://www.playvectrex.com/designit/chrissalo/appendixa.htm#12 (accessed 16 March 2010).

Salomon, Chris. ‘An Introduction to Vectrex Programming: Sound Playing’. http://www.playvectrex.com/designit/chrissalo/soundplaying.htm (accessed 16 March 2010).

Salomon, Chris. ‘Appendix C: The 6522A Chip From The Vectrex Perspective’. http://www.playvectrex.com/designit/chrissalo/via3.htm (accessed 16 March 2010).

SEGA AM7 Team. *Streets of Rage 3*. Software for SEGA Mega Drive / Genesis. 1994.

Semenov, Alex. TFM Music Maker. Software for Windows OS. 2009 ftp://ftp.untergrund.net/users/shiru/tfmmaker14.zip (accessed 23 March 2010).

Sepchat, Alexis; Descarpentries, Simon; Monmarché, Nicolas and Slimane, Mohamed. ‘MP3 Players and Audio Games: An Alternative to Portable Video Games Consoles for Visually Impaired Players’, in *Computers Helping People with Special Needs: 11th International Conference, ICCHP 2008.* Linz, Austria, July 9-11, 2008. Berlin: Springer Berlin, 2008.

Slocum, Paul. ‘Atari 2600 Sequencer Kit.’ http://www.qotile.net/sequencer.html (accessed 16 March 2010).

Slocum, Paul. Loopcart. Software for Atari 2600. http://www.qotile.net/loopcart.html. 2004 (accessed 16 March 2010).

Slocum, Paul. Synthcart. Software for Atari 2600. Published by AtariAge.com. 2002. http://www.qotile.net/synth.html (accessed 15 March 2010).

Sonic Team and SEGA Technical Institute. Sonic the Hedgehog 2. Software for SEGA Mega Drive / Genesis. 1992.

Staugaard, Andrew. *6809 Microcomputer Programming and Interfacing, With Experiments*. Indianapolis: Howard W. Sans, 1981.

Texas Instruments Semiconductor Group. ‘SN76489 Datasheet’. http://soundprogramming.net/electronics_reference/manuals/SN76489datasheet.pdf (accessed 15 March 2010).

Viens, David. Chipsounds. Software for Mac OS X and Windows. Published by Plogue. http://www.plogue.com/?page_id=43 (accessed 15 March 2010).

Wittchow, Oliver, Nanoloop. Software for Game Boy. http://nanoloop.com/node/13 (accessed 15 March 2010).

Wolf, Mark. ‘Imaging Technologies’, in *The Medium of the Video Game*. Austin: University of Texas Press, 2002.

Wolf, Mark. *The Video Game Explosion: A History from PONG to Playstation and Beyond.* Santa Barbara: Greenwood Publishing, 2007.

Yamaha Corporation. YM2612 Application Manual: Overview. http://www.smspower.org/maxim/Documents/YM2612#overview (accessed 23 March 2010).

Yamaha Corporation. YM2612 Application Manual: Part I: Memory Map. http://www.smspower.org/maxim/Documents/YM2612#partimemorymap (accessed 16 March 2010).

Yarusso, Albert. ‘Atari Age - Have You Played Atari Today?’ http://www.atariage.com/ (accessed 15 March 2010).