

Audio Time Compansion for Studio and Performance Synchronization

by

Marc Bayard LoCascio

SUBMITTED TO THE DEPARTMENT OF ARCHITECTURE
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF THE DEGREE

MASTER OF SCIENCE IN VISUAL STUDIES

AT THE

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

SEPTEMBER 1987

©Massachusetts Institute of Technology 1987

Signature of the author

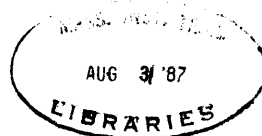
Marc LoCascio
Department of Architecture
August 14, 1987

Certified by

Barry Vercoe
Professor of Music and Technology
Thesis Supervisor

Accepted by

Nicholas Negroponte
Chairman
Departmental Committee for Graduate Students



Rotch

Audio Time Compansion for Studio and Performance Synchronization

by

Marc LoCascio

Submitted to the Department of Architecture on August 14, 1987
in partial fulfillment of the requirements of the degree of
Master of Science in Visual Studies

ABSTRACT

Recent research has provided systems that play musical accompaniment on real-time synthesizers while following the performance of human soloists. Another solution to electronic music's problem of syncing live players to pre-recorded tapes is a system using a time code synchronized tape deck whose audio output is pitch corrected for any tape speed deviation. A frequency shifter with SMPTE time code input is required for this task. With a computer-based, variable-rate time code generator, a musical performance can be compared against an internal representation of its score and external devices—be they audio or video tape machines, video disk players, or MIDI synthesizers—synchronized to the live action. Two foreseeable uses of this interactive time code generator will be the freeing of musical sequencing tasks from the host computer in "synthetic performer"—allowing the computer to concentrate on pitch tracking and score following while sequencing and synthesis is accomplished by dedicated MIDI machines—and the synchronization of visual images—such as from video disk—to live musical performance. The complete system of time code generator and reader/pitch shifter will be outlined in this paper, and other uses for the technology developed therein will be suggested.

Thesis Supervisor: Barry Vercoe
Title: Professor of Music and Technology

This work was supported by a grant from the System Development Foundation.

Table of Contents

1. Introduction	1
1.1 Project Motivations	1
1.2 The Live Music to Video Connection	1
2. The Hardware	3
2.1 The Computer Music Workstation	3
2.2 The Time Code Reader	4
2.2.1 SMPTE Time Code	4
2.2.2 The Otari I-0055 Time Code Reader IC	4
2.2.3 The IEEE-488 Bus and the Motorola MC68488	6
2.2.4 The Motorola MC6809	7
2.3 The Time Code Generator	8
2.3.1 Vari-speed Clocks	8
2.4 Audio Sources and Synchronizers	8
3. The Software	11
3.1 Pitch Shifting	11
3.1.1 Zero Crossing	12
3.1.2 Constant Window Size	12
3.1.3 Variable Window Size with Zero Detection	13
3.1.4 Pitch Tracking Techniques for Window Estimation	13
3.1.5 Envelope Shapes for Window Scaling	14
3.2 Time Compansion	15
3.2.1 Rotating Tape Head	15
3.2.2 Analogies to Pitch Shifting	16
3.3 Alternate Methods	18
3.3.1 Phase Vocoder	18
3.3.2 Marked Soundfiles	19
4. Integration and Applications	20
4.1 Real-time Control	20
4.1.1 Software for Time Code Parsing and Feedback to the Pitch Shifter	20
4.2 The Audio/Visual Studio System	21
4.2.1 Csound Modules	21
4.2.2 A Constant Duration Sampling Keyboard	23
4.2.3 Bobcat RS422 Remotes	22
4.2.4 A Disk Based Synthesis/Recording/Editing System	23
4.3 The Multi-media Performance System	23
4.3.1 The MIDI to SMPTE Connection	24
4.3.2 The Random Access Video-disk System	24
4.3.3 DSP Hardware for Real-time Systems: The Future is Soon	26
5. Acknowledgements	27
Appendix A: Hardware	29
Appendix B: Software	33
Bibliography	46

1. Introduction

1.1. Project Motivations

The master versus slave issue has long been a problem in audio and video synchronization. In the video production process, audio tape machines are usually slaved to video machines due to the relative simplicity of their servo mechanisms. Similarly, in MIDI sequencer to tape synchronization, MIDI devices are inevitably slaved to SMPTE time code from an audio tape. In live performance, human musicians until recently have been slaved to their MIDI and tape counterparts. With the coming of real-time computer score followers—generating time bases for synthesizer accompaniment to live performers—we finally see the tables turning on the human to MIDI sync problem [Vercoe]. The human to audio tape issue is more difficult to solve, however. Since analog audio tapes would have to be speeded up or slowed down to follow a live performance, a pitch shifter must be devised which can track the tape's speed and provide the appropriate pitch compensation. This is the goal of this research. Also, some concept of non-real-time, or "virtual," SMPTE time code must be realized—a representation of a relative time in a score as opposed to a strict clock time. Such a time code generator connected to a computer music workstation running score following software can provide a flexible time stream for the synchronization of audio tapes or even video (such as from video disk) for the accompaniment of music.

1.2. The Live Music to Video Connection

Another inspiration for the implementation of virtual time code comes from visual artists. Filmmaker Richard Leacock and his students at the MIT Media Lab's Film/Video group expressed an interest in using the technology of pitch tracking and score following to synchronize video imagery to live performance in the fashion of Prof. Leacock's earlier work with the Metropolitan Opera Company.

Since it is possible to sync synthesizers to human players, it should be possible to sync video disk imagery as well. A microcomputer based system using SMPTE time code readers and random access video disk players has already been implemented at the Film/Video group [Sasnett]—a virtual time code generator interlocked to a score following routine is all that is needed to complete the performance system.

2. The Hardware

2.1. The Computer Music Workstation

The basic system for this project consists of a Hewlett-Packard Series 300 "Bobcat" computer equipped with custom built analog-digital convertors, MIDI interface, and SMPTE time code reader/generator.

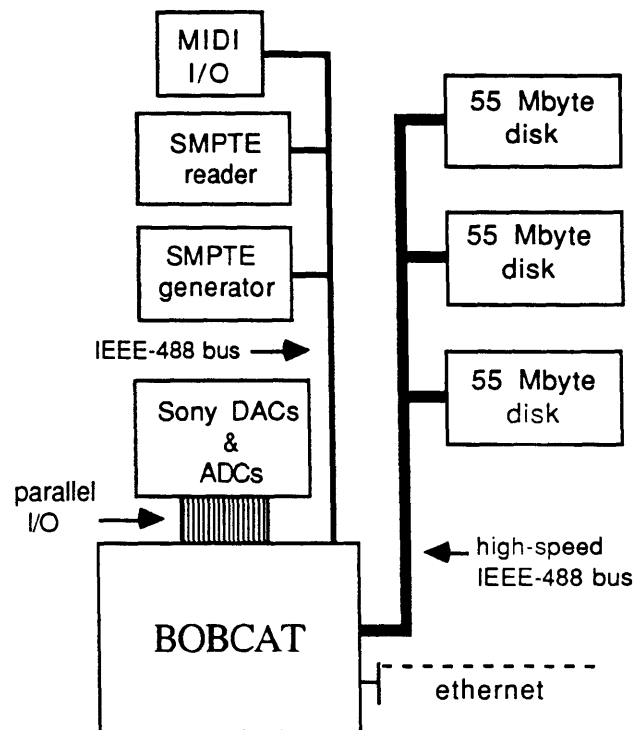


Figure 1: The Media Lab computer music workstation

The Bobcat is a microcomputer based on a Motorola 68020 processor with a 68881 math co-processor and 6 Mbytes of RAM, running an HP version of Bell Labs' System V UNIX with 4.2 BSD enhancements. Three 55.5 Mbyte Winchester disks on a high speed disk interface are used for sound and data storage. Four of the HP machines are on an ethernet link with the music studio's DEC VAX 11/750, VAXStation II, and Sun III—all running UNIX and the Csound music processing language. The ADC's and DAC's are 16 bit Sony convertors with a 48 kilohertz

sampling rate, and MIDI I/O is handled by a Roland MPU-401 MIDI processing unit (all connected to the Bobcats by in-house adaptor cards). Soundfiles are stored to disk as regular UNIX files for Csound software synthesis and other analysis/synthesis methods. For sound input and output, however, scratch soundfiles must be created on a raw partition of a blank disk in order to achieve the data throughput necessary for 48 kilohertz stereo sound. The time code reader and generator construction is part of this project and, as such, will be presented in some detail.

2.2. The Time Code Reader

2.2.1. SMPTE Time Code

SMPTE time code was adopted by the Society of Motion Picture and Television Engineers in 1981 as a standard for audio and video time stamping analogous to the frame and footage counts used for synchronization in film production. The time code signal consists of 80 bits of serial information per frame at a frame rate of 24, 25, 29.97, or 30 frames per second (for film, European video, NTSC color video, or black and white video, respectively). The bits are "bi-phase" encoded—meaning that each bit cell of the signal is denoted as a "one" by a level transition or as a "zero" by no transition. Thus, a 30 frame per second stream of all zeroes would produce a 1200 Hz square wave while a stream of all ones would produce a 2400 Hz square wave. The actual hour, minute, second, frame, and user definible bit assignment for one frame is shown in figure 2.

2.2.2. The Otari I-0055 Time Code Reader IC

The Otari I-0055 time code reader IC, used on our reader card, takes a conditioned (TTL compatible) time code signal and outputs its component parts—BCD (binary coded decimal) hours, minutes, seconds, and frames data—when requested

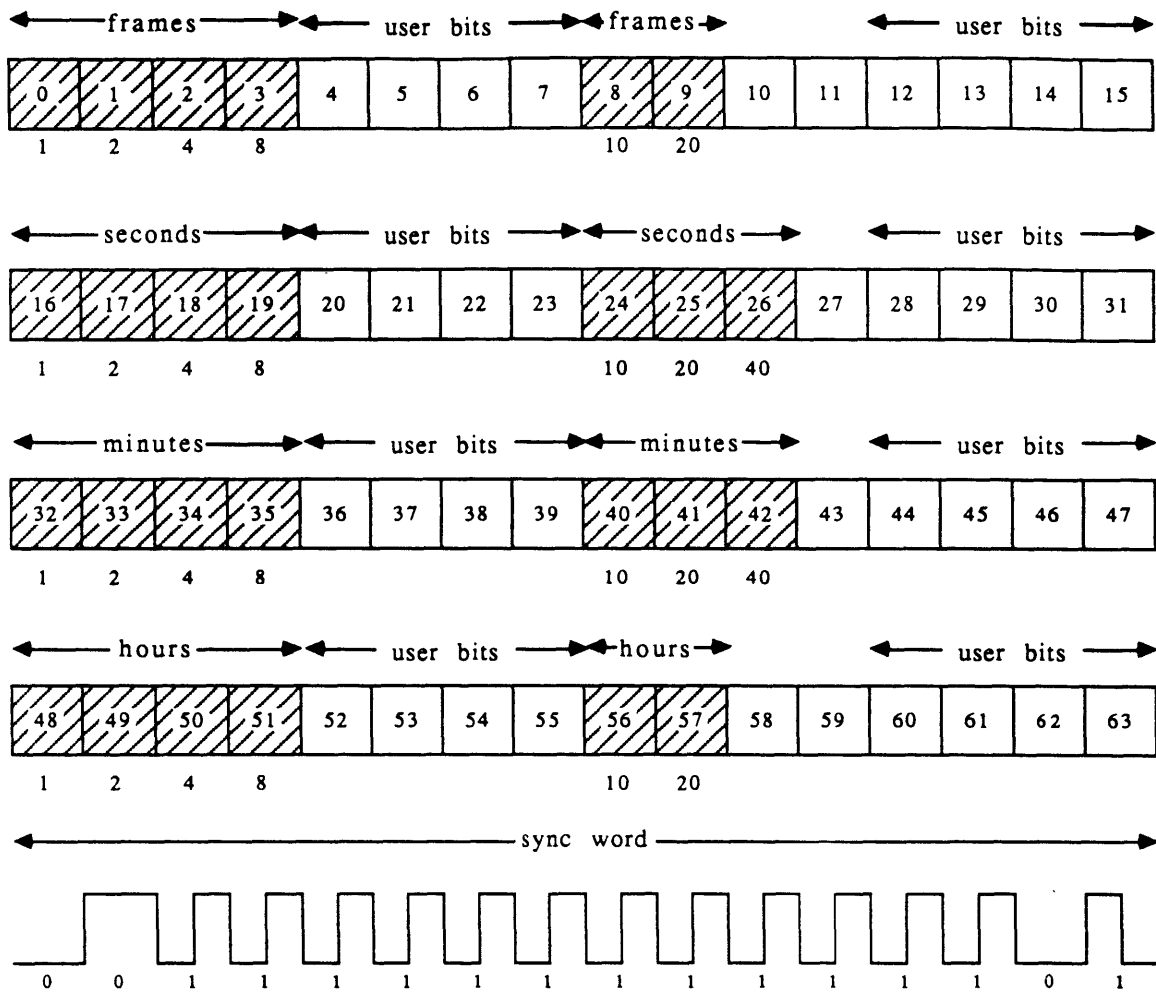


Figure 2: SMPTE Time Code Format and Bit Assignment

by its host microprocessor. Also provided by the reader chip are a sync signal for acknowledgement of each frame received and a SMPTE clock signal corresponding to the 1200 Hz bit cell transitions of the time code. A pulse train generated by feeding this output clock through a pair of monostables can be counted alongside the output of a 2400 Hz crystal based clock in order to judge the actual speed and sub-frame position of the time code with extended accuracy. A detailed schematic of this mechanism can be seen in figure A.1 in the hardware appendix to this paper. Since the time information is stored in the SMPTE code as BCD data, a table look-up step is needed to convert the time stamp into UNIX readable form. The BCD to

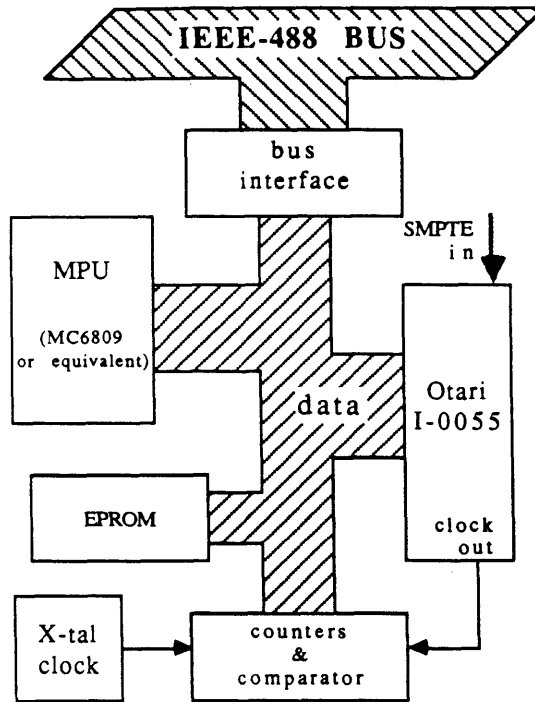


Figure 3: Time code reader block diagram

binary conversion table is burned into the same EPROM as the processor program memory. A Motorola MC6809 microprocessor is used to coordinate this and other functions of the time code reader card.

2.2.3. The IEEE-488 Bus and the Motorola MC68488

The IEEE-488 bus is a byte serial communications link intended primarily for instrumentation uses. Since two such busses are included with every Bobcat (a high speed disk bus and a general purpose I/O bus)—each capable of supporting 16 individually addressed devices—this was the best choice for the interface to the SMPTE reader. A special chip set from Motorola—the MC68488 general purpose interface adaptor and 3447 bus transceivers—supports the IEEE-488 bus protocol, simplifying the task of interfacing peripherals to this complicated bus structure. The bus address of each card, as well as the reader's operating mode—basic timecode,

timecode plus subframe, speed only, user bits only, or software controllable—can be selected by the user by setting DIP switches on the card. The controller chip handles the bus handshaking and provides registers for data transmission and reception, address designation, bus status, and the like (specified fully in Appendix *). Therefore, the MC6809 processor can easily poll the bus and supply time code information to the Bobcat as requested. This proves to be a most elegant design.

2.2.4. The Motorola MC6809

The MC6809 is the newest of Motorola's 6800 family of microprocessors, and it supports a wide variety of addressing modes. This simplifies the communication between the numerous devices on the time code reader card. The individual modules (shown in detail in hardware appendix figure A.1) are the IEEE-488 bus interface, the time code reader IC, the EPROM for program and BCD look-up table storage, sub frame counter, speed counter, and address switches. Information can be passed among these modules—usually from some register to the output register of the bus controller for transmission to the Bobcat—by the MC6809. The assembler code initializes the MC68488 and then checks the DIP switches for the card's operating mode. Depending on the mode, the processor either stuffs the requested information onto the IEEE-488 bus as quickly as the Bobcat will accept it or waits until the Bobcat sends a request byte to the reader. The former modes are more efficient for fast control applications whereas the latter mode allows the user to select different types of data as needed by his or her software. Maps of all operating modes, the associated DIP switch settings, and data transmission formats are given in the appendices. It is therefore imperative that the user check the operating mode switches before interpreting data received from the SMPTE reader. In-depth schematics and maps of the MC6809 address space and reader operating modes are given as figures in the hardware appendix, while the commented MC6809 assem-

bler code appears in the software appendix.

2.3. The Time Code Generator

The proposed time code generator is MC6809 microprocessor based and has the capability of producing variable-speed time code for the purpose of synchronizing external devices to the score matched time of a live performance. The SMPTE code has a virtual format of 30 frames per second—computed and converted to BCD data by the MC6809. The clock which controls the rate at which the time code is shifted out of the holding registers has an analog bias with a feedback loop to regulate the output speed. The workstation passes down a time compression or expansion (i.e. compansion) factor to the SMPTE generator, which alters its clock rate accordingly. The success with which the virtual time code can be followed depends very much on the capabilities of the reader to which the time code is sent.

2.3.1. Vari-speed Clocks

The module which sets this SMPTE time code generator apart from others presently on the market is the variable-speed analog clock which determines the actual output speed of the time code. The analog clock output, using a feedback loop in which it is compared to a crystal based clock, controls the rate at which the time code frame's bits (computed by the microprocessor and loaded into a register after every 80 clock ticks) are shifted out of the holding register. Eight bits of time offset are sent to the generator by the Bobcat, and these are compared against the actual frequency of the vari-speed clock as determined in comparison to the crystal clock. The vari-speed clock is then adjusted accordingly.

2.4. Audio Sources and Synchronizers

There are two audio domain recipients of our virtual time code—Sony professional tape decks with integral chase lock synchronizers and the Roland SBX-80

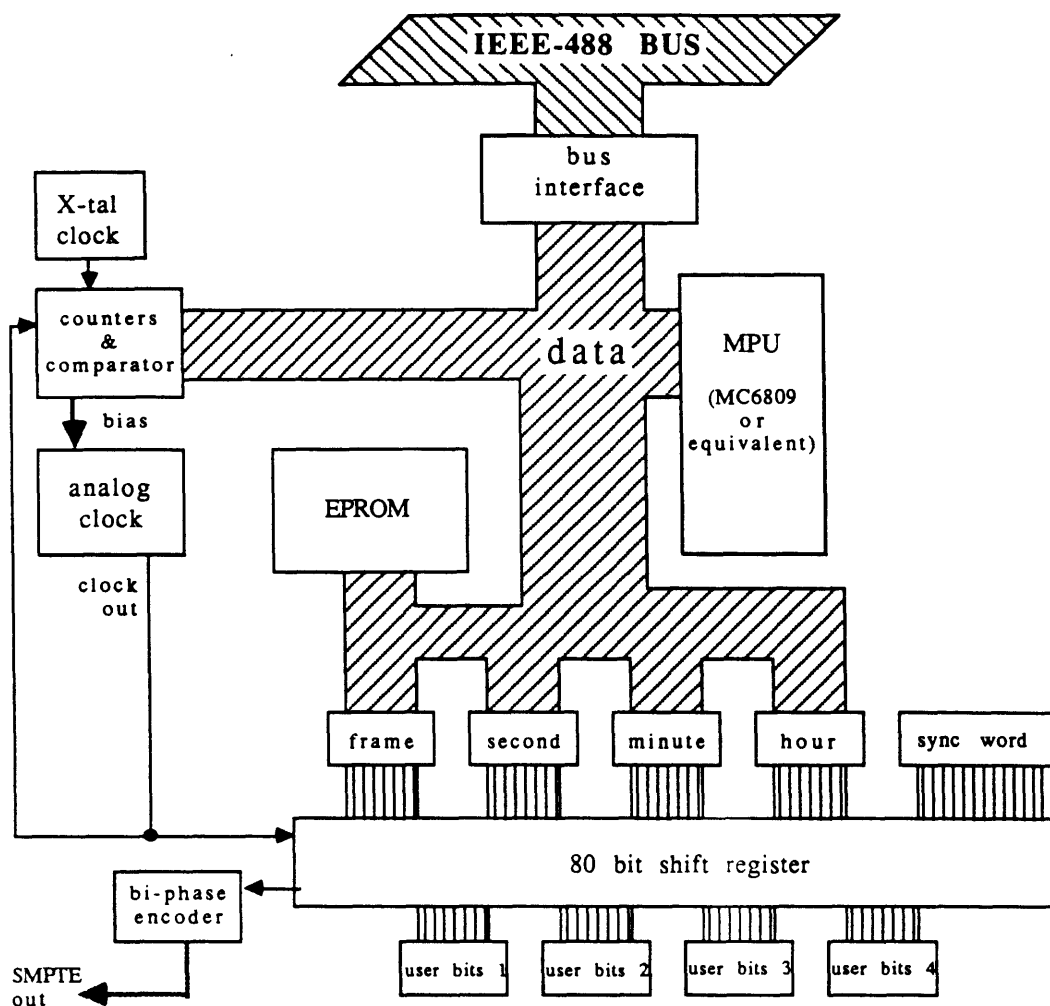


Figure 4: Time code generator block diagram

SMPTE to MIDI sync box. The Sony machines are the APR-5003 1/4" half track analog tape deck with center track time code and the PCM-3202 two track DASH (digital audio stationary head) recorder. The synchronizers in these machines provide full chase mode interlock but are susceptible to time code discontinuities (thus precluding virtual time code with added or dropped frames and a fixed clock rate). The DASH machines can track a plus or minus 15% speed variation before muting their output, while the analog machine will follow a much wider time variation. Both machines have rapid but hardly instantaneous transport servos. For this reason, the pitch shifter SMPTE input must read the time code from the audio tape

itself instead of the master time code which the tape deck is chasing. Fortunately, both decks allow this. The Roland sync box follows approximately the same SMPTE code speed variation as the DASH machines and provides full chase mode synchronization to sequencers which respond correctly to MIDI song pointer, clock, and start/stop commands. It should be noted that the SBX-80's function could be served by a Bobcat with a SMPTE reader card and an MPU-401 for MIDI I/O. The modularity of the computer music workstation alleviates the need for costly special purpose devices.

3. The Software

3.1. Pitch Shifting

Simple audio time variation can be achieved with a variable speed tape deck or by playback of a digital soundfile at different sampling rates, and pitch shifting algorithms can be used to compensate for the subsequent pitch fluctuations. Sound segments are repeated or stretched using a variable speed phase accumulator to determine the index into the look-up table of samples—with the resulting resampling providing the pitch change.

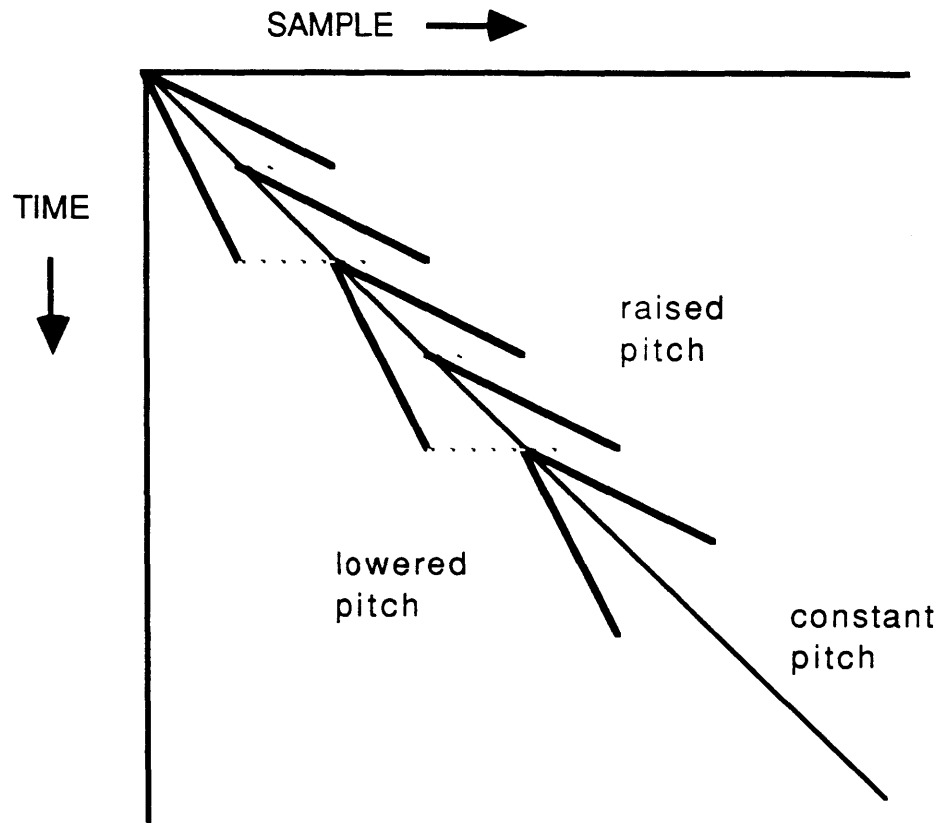


Figure 5: Time vs. sample for pitch shifting

For the best audio quality, loops between windows should be synchronized to the period of the pitch being shifted to reduce amplitude modulation problems. A

variety of window sizes and shapes—as well as schemes for overlapping of multiple windows—have been implemented during the course of this research to determine their subjective fidelity.

3.1.1. Zero Crossings

A simple and somewhat naive algorithm for pitch shifting loops adjacent windows on zero crossings with a brief cross-fade (perhaps 16 samples). While this technique provides good results with simple signals, sounds as complex as speech or bowed strings take on a raspy, metallic quality due to the over emphasis of upper partials. Since simple zero crossings are oblivious to the period of the signal, higher order harmonics are used as looping points more often than the fundamental (assuming more than two prominent harmonics). The absence of overlapping windows removes redundant information which would help to smooth the harsh timbre of the processed sound.

3.1.2. Constant Window Size

Another simple—yet more elegant—algorithm uses overlapping windows of fixed size. The windows should be along the order of 4096 samples at a 48 kilohertz sampling rate (a sub-audio period). Each window is scaled by a sinusoidal envelope and is added to a second window 90 degrees out of phase with itself. With sine squared envelopes, the output has constant power at all times, providing a seamless cross fade. While this technique is vastly superior to the zero crossings method for complex signals, simple signals such as sine waves suffer from amplitude modulation when the window size is not a multiple of the period of the audio signal. The contrast in effectiveness of these two simple algorithms on material of different complexity suggests that some adaptive technique which can vary window size according to the audio source should be devised.

3.1.3. Variable Window Size with Zero Detection

The first and most basic adaptive algorithm written uses 50% overlapping windows that are phase aligned at zero crossings of the source signal. When one window reaches its maximum amplitude and the other has faded out, zero crossings are sought among both sets of input samples, and the cross fade does not begin until the common points are found. The envelopes thus take the form of a sine wave rising to full amplitude, sticking at full amplitude until a zero crossing is found, and only then decreasing to zero amplitude. The drawback to this algorithm is the same as that for simple zero crossing looping—namely, upper partials are emphasized in complex signals. As a result, this method proves rather inferior to the simple, fixed period window technique for most input signals.

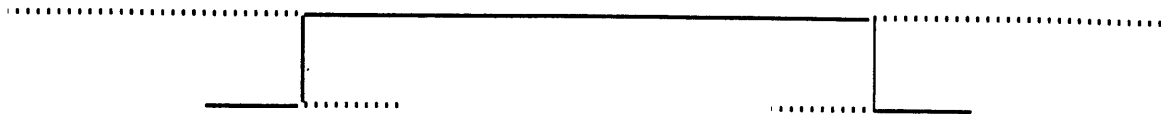
3.1.4. Pitch Tracking Techniques for Window Estimation

A further improvement on these pitch shifting algorithms uses pitch tracking techniques for the estimation of optimal window size. At this degree of complexity we have left the realm of real time on general purpose workstation hardware. Simple peak detection can be used—as it is the simplest general period estimation method—but octave errors still contribute to a harsh, metallic sound not unlike that of algorithms using zero detection techniques. A simple improvement on the result generated by this technique is to assume that an octave error has in fact occurred and that the determined wavelength is half of the actual λ , taking twice the result as the wavelength used. A better (but still relatively cheap computationally) pitch tracking technique is that of calculating wavelengths using delayed differences. In this method, two pointers into a soundfile scan along in sync some number of samples apart as long as the difference between the samples to which they point is within some tolerable margin (referenced to an accumulated power signal for the soundfile). When the difference between the two samples exceeds the error margin,

the leading pointer scans ahead alone until the sample values are again within tolerance—at which time the pointers again proceed together. After some number of samples have been traversed with the two pointers in sync, the number of samples between the two pointers is taken to be a valid wavelength for the signal (lambda value) and a frequency can be calculated given this value and the sampling rate of the soundfile. Again, twice the value of the given lambda can be used as a safeguard against octave errors. Since window size is fairly flexible for the pitch shifting algorithm, these pitch tracking techniques can be used to determine the size of the windows at run time. The constant difference technique also has the beneficial side effect of not allowing looping on high frequency transients of the audio signal (if this can indeed be avoided). Minimum and maximum lambda values are defined in the pitch tracking routine, and frequencies outside of this range will not provide looping points to the pitch shifter. Some average window size must be used for noisy signals in which no periodic window is ever returned, but our experience with constant window size pitch shifters shows us that the size is not very critical in these cases.

3.1.5. Envelope Shapes for Window Scaling

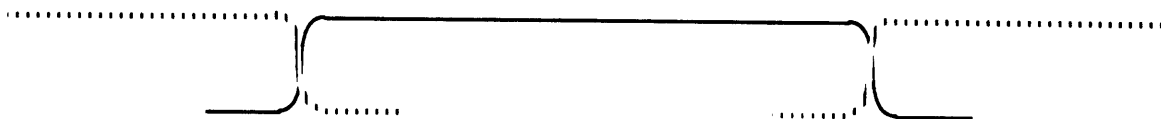
Our final concern in the creation of pitch shifting windows is the shape of the envelopes with which the looped windows will be scaled. The three types that we have seen are rectangular windows with butt-spliced looping, 50% overlapping windows with sine squared crossfade, and sine squared crossfades between steady-state, unscaled windows. Of the three techniques, the last is theoretically the best, assuming that adequate looping points can be found. Constantly overlapping windows produce unwanted flanging or phase shifting effects on the audio signal; non-overlapped, steady-state windows with fast crossfades would alleviate this problem. Rectangular windows would be acceptable if glitch-free looping points could be



rectangular windows



sine squared windows



rectangular windows with sine squared cross-fades

Figure 6: Envelope Shapes for Window Scaling

found, but even the best pitch tracking techniques can not always guarantee this for us.

3.2. Time Compansion

3.2.1. Rotating Tape Heads

The earliest attempts at constant-pitch time compansion used audio recorders with rotating tape heads [Lee]. Four heads mounted on a spinning drum around which the audio tape was wrapped effectively created windows into the soundstream which were either looped or discarded depending on whether the sound was to be

expanded or compressed. Essentially the same technique can be used in the digital domain for time manipulation. In the crudest manner, rectangular windows with fixed period can be taken from the audio signal and looped or discarded accordingly.

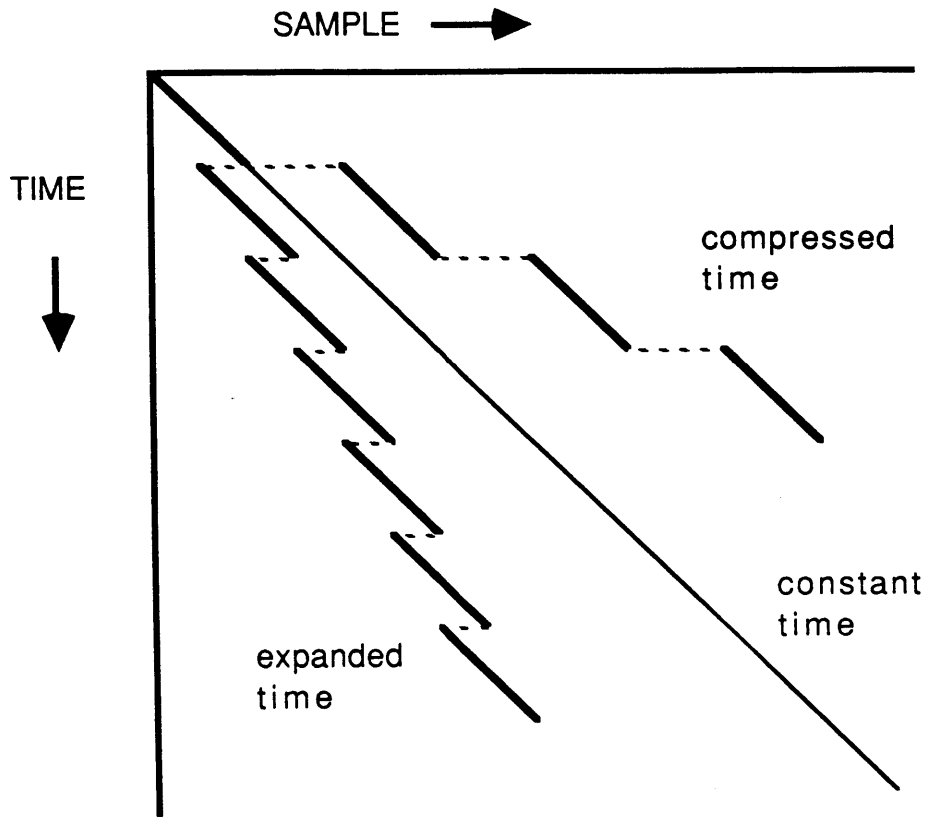


Figure 7: Time vs. sample for time compansion

Figure 7 shows the basic time versus sample index output relationship. Improvements on this technique consist of window looping and overlapping schemes like those of the pitch shifting algorithms to smooth discontinuities at window boundaries.

3.2.2. Analogies to Pitch Shifting

The time compression techniques are obviously quite similar to those of pitch shifting, as the time versus sample output graphs indicate. Also, as previously men-

tioned, time compansion can be achieved simply by altering audio tape speed or digital sample rate playback and then pitch correcting the audio signal. Simple examples of time compression and pitch lowering with 50% overlapping sinusoidal windows are given as figures 8a and 8b.

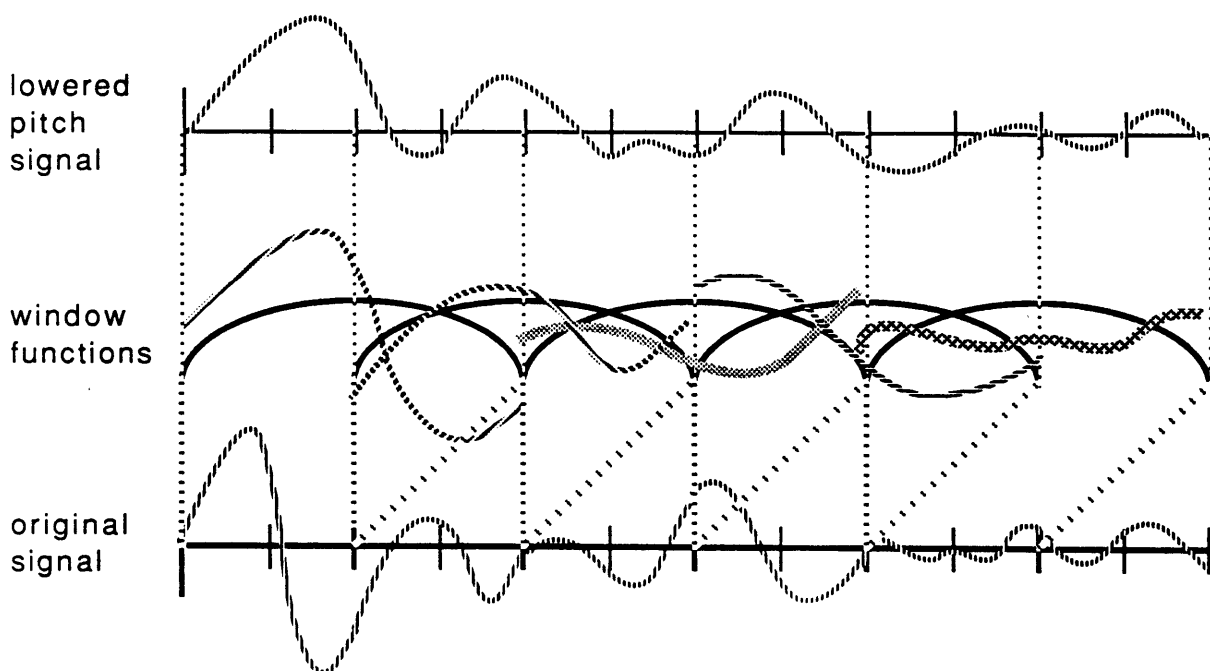


Figure 8a: Simple Pitch Lowering with Sinusoidal Windows

Thus, in the recording studio, a pitch shifter and an audio tape recorder comprise the basic time compander. The addition of time code feedback from the tape deck to the pitch shifter is a great convenience in this task, since pitch correction can be handled automatically instead of calculated by the recording engineer from tape speed deviation and cents of pitch shift. A pitch shifter with SMPTE inputs for this purpose is available commercially (the Lexicon 2400 Stereo Audio Time Compressor/Expander), but a modular, general purpose solution to the problem using a computer capable of other functions is of considerable advantage to the cost-effective studio.

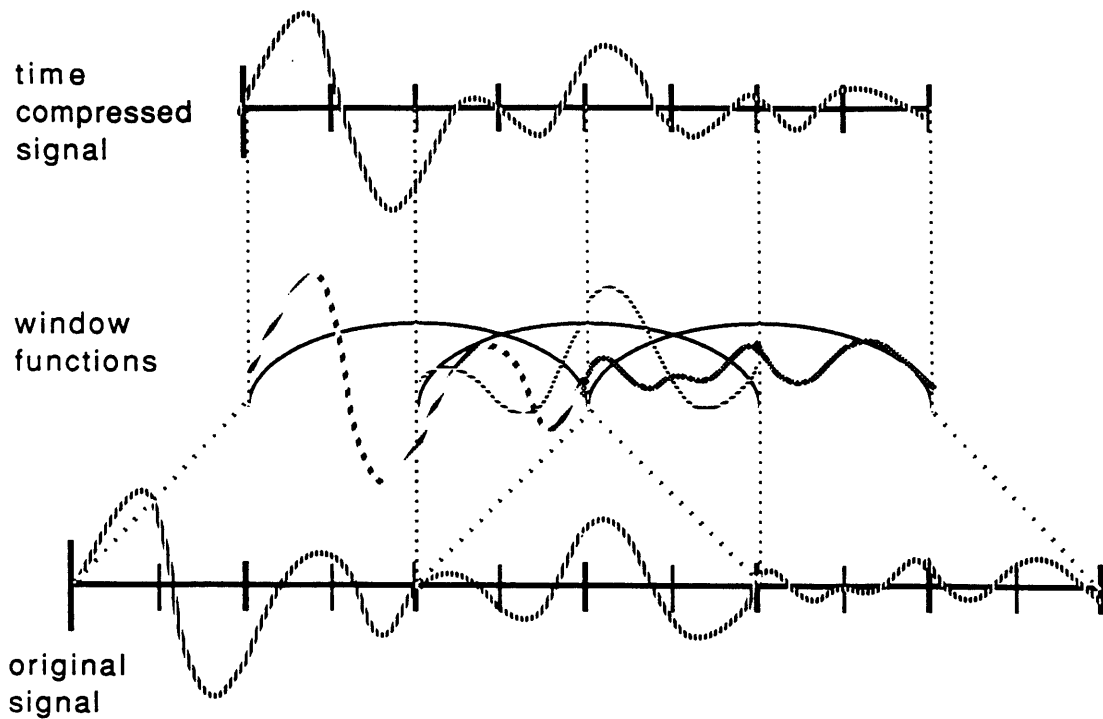


Figure 8b: Simple Time Compression with Sinusoidal Windows

3.3. Alternate Methods

A variety of other techniques exists for the high fidelity time and pitch manipulation of sound, primarily outside of real-time. Audio analysis and resynthesis methods, such as the UCSD phase vocoder [Dolson], are capable of very "musical" signal processing, but they also require significant computing power and user input as to parameter definition. A new idea for high quality audio manipulation purely in the digital time domain has also been discussed.

3.3.1. The Phase Vocoder

The phase vocoder differs from the previous time and pitch manipulation techniques in that it is in fact an analysis/synthesis technique. Sound processed by the phase vocoder is analyzed (in non-realtime) and resynthesized using parameters derived from the analysis. It is also a frequency domain operation. Frames of filter

banks changing over time are decoupled from their drive functions (the oscillators). In this manner, software oscillators can be applied to the resultant filter banks sweeping more slowly through their time slices than they originally occurred to produce time expansion, or higher pitched drive frequencies can be applied to the filters to affect pitch shifting with constant time. Of course, this is a computation intensive task and can not be achieved in realtime in a typical studio.

3.3.2. Marked Soundfiles

Yet another non-realtime technique which can be applied to computer soundfiles for time and pitch manipulation would be an audio analysis and marking process which denotes optimal looping points and window types for the particular sound segments. In this theoretical method, time or frequency domain techniques could be used to select unambiguous, periodic looping points—preserving attacks and other transients while looping on relatively pitched steady-state segments. This technique is vaguely similar to the "Linear Arithmetic" synthesis used by the Roland Corporation in their new, critically acclaimed D-50 synthesizer. LA synthesis uses PCM encoded attack transients mixed with wavetable look-up oscillators and/or looped PCM sounds for the steady-state portion of the sound. The results are amazingly lifelike in a cost-effective keyboard. Using a similar process for the time variant reproduction of polyphonic music is obviously much more difficult but would be feasible on a DSP processor equipped music workstation. Prerecorded or computed soundfiles on disk can be accompanied by a control file containing a table of sample numbers between which looping can be performed with minimal signal degradation. This table can be examined whenever the playback of the file is found to be out of sync with the live performance, and an appropriate compansion breakpoint can be located. This technique would have many uses on a disk based synthesis/recording/editing workstation, as will be discussed later.

4. Integration and Applications

4.1. Real-time Control

As UNIX is notoriously bad for real-time development, it is arguably best to keep to a minimum one's expectations of the workstation itself and build as much power as possible into the hardware peripherals. In the IEEE-488 time code reader, for instance, BCD to binary data conversion and crystal clock tick to speed calculations are performed by the MC6809 microprocessor, not the Bobcat. The problems associated with real-time control have been addressed elsewhere [Puckette], so a cursory discussion of the specific factors in this application should suffice. The process scheduler must respond as quickly as possible to messages signifying changes in tape speed from the SMPTE reader and pass requests for changes of virtual performance time from the score follower to the SMPTE generator. Tape speed variation, when detected, causes a proportional change to the phaser frequency at which samples are read from the look-up tables. Using real-time process priority, as well as memory locking and optimized C or 68020 assembler programming, we can achieve adequate timer accuracy and audio processing power for one channel of simple pitch correction (as a test case). In the long run, the system should be re-implemented with a high-speed DSP board to handle the number crunching for the host computer.

4.1.1. Software for Time Code Parsing and Feedback to the Pitch Shifter

In the simple case of using the time code reader to determine the speed of tape travel in order to pitch correct audio, the reader card can be used in the speed detection only mode. In this mode, the reader reports only speed information—derived from the SMPTE clock compared against the 2400 Hz crystal clock—passing it on to the Bobcat whenever its address is read on the IEEE-488 bus. The value returned is 80 (for the number of bit cells per frame of time code) minus the

number of crystal clock ticks accumulated between the sync pulses (signifying the beginning and ending of the read time code frame). In turn, the Bobcat pitch shifting software polls the reader 30 times a second and interpolates the sample table look-up increment between the values for the last two reported speeds. Since this modulation rate is in the low audio frequency range, drastic pitch changes should be fairly inobtrusive—being of too high a frequency to be heard as vibrato and of too low a pitch to produce objectionable FM sidebands.

4.2. The Audio/Visual Studio System

The music workstation—equipped with the hardware and software modules previously described—is the center of a network of machines configurable in any number of ways for audio and video production. Virtual remotes for tape deck control can be implemented in mouse-driven computer graphics with an RS422 physical link. Direct (i.e. digital) data links can be established between digital audio tape machines, samplers, and workstations. SMPTE time code can be used for MIDI, audio, or video synchronization. Various computers of different types can be connected by a combination of ethernet, MIDI, and RS232/RS422 data transmission. Ultimately, this standardization and modularity provides a vast array of new tools for the musician at the price of moderate software development.

4.2.1. Csound Modules

One immediate application of the signal processing end of this research is pitch shifting and time compression modules for the Csound software synthesis language. Csound is a C and UNIX based language which is presently running on the VAX 11/750, VAXStation II, and Bobcat workstations at the MIT Media Laboratory. It is a powerful software package consisting of modules for audio synthesis, analysis, and processing. The addition of pitch and time manipulation operators would allow composers to create constant duration samplers and would allow the

time processing of sound effects, foleys, and the like for film and video post-production. This music language is primarily for non-real-time applications on larger machines, but, with the escalating capabilities of small and cheap workstations, real-time uses will soon be feasible.

4.2.2. A Constant Duration Sampling Keyboard

The marriage of a high powered workstation with on-line sound storage, MIDI keyboard input, and a DSP processing unit would allow the creation of a real-time, constant duration sampling keyboard with many musical applications. Samplers presently on the market affect a pitch change simply by re-playing the recorded sample at a higher or lower clock rate. The resulting sound is appropriately shorter or longer in duration—altering any rhythmic content of the sample. On the constant duration sampler, rhythm loops of definible pitch and duration could be layered into complex polyrhythmic patterns—a present impossibility. Sound effects could be stretched, looped, pitch shifted, and dropped into a soundtrack under precise SMPTE control. Pre-recorded dialogue could be relatively easily dubbed to fit existing film scenes. The possibilities of such a keyboard on a computer music workstation are many and exciting.

4.2.3. Bobcat RS422 Remotes

Another interface which will be of immense value in the computer automation of the audio/video production studio of the future is the RS422 remote control of audio and video tape transports. A standard proposed by the SMPTE organization calls for the ability to control all studio control functions of tape decks, mixers, switchers, and the like via an addressed network of serial links. This would allow the workstation running audio manipulation software not only to process signals emanating from the audio decks but to control their record, play, and shuttle functions. A window based virtual studio consisting of icons for signal processing units,

analog and digital audio sources, and mixing facilities can be drawn by the audio engineer on his or her computer screen and implemented by the workstation with no further operator involvement. The RS422 remotes will replace the old contact-closure remotes—typical on present tape machines—with standardized software control. This will lower the price of studio automation systems and encourage development of high end equipment for the smaller studio market.

4.2.4. A Disk Based Synthesis/Recording/Editing System

The ultimate application of this technology in the studio would be the creation of a disk based audio synthesis/recording/editing system for multi-media audio production. Similar to the LucasFilm/DroidWorks Sound Droid, this system would consist of a general purpose computer with many peripheral modules linked together in a sensible and expandable network for audio production. Time code readers and generators, along with RS422 remotes, can be used to control tape machines and mixers, while MIDI links can be used to control outboard effects processors, synthesizers, samplers, and other mixers. A great advantage to this system would be the on-line storage of sound files. Large magnetic disks can be used for audio recording and rapid data access, while optical data disks can be used for long term archiving—such as for sound effects libraries or audio master recordings. All processing onboard the workstation or on auxiliary workstations will be tightly synchronized to all other studio processes via SMPTE time code. This modularity will allow very powerful studio configurations to be realized at steadily decreasing costs.

4.3. The Multi-media Performance System

The final application for this research in the real-time realm is an audio/visual performance system in which the human artist acts as master to the computer accompanist. The virtual time code generated in response to live performers can be used to drive any number of audio, video, or MIDI slaves. In turn, SMPTE

equipped audio decks can control pitch shifting processes on A-to-D-to-A converter equipped workstations. With such an open ended system (as is portrayed in part in figure 9), the creative possibilities are limited only by the imagination and talent of the user.

4.3.1. The MIDI to SMPTE Connection

There are a number of commercially available boxes which convert SMPTE time code into MIDI clock signals for sequencer synchronization—among them, the Roland SBX-80, the Southworth Jam Box, the Fostex Model 4050, and the Synchronous Technologies SMPL System. An equally useful but heretofore unimplemented box, though, is a MIDI to SMPTE sync box, which would synchronize audio or video source material to live musicians. This is possible with the combination of a workstation with MIDI input, score following software, and a virtual time code generator. The first two thirds of this chain are components of Barry Vercoe's "Synthetic Performer" system. All that is needed is the time code generator hardware and the replacement of the MIDI accompaniment routine of the performer software with a virtual time offset generator. As has been previously stated, this virtual SMPTE code can be fed to any number of SMPTE synchronizable devices, including audio and video tape decks, SMPTE to MIDI synchronizers, or videodisk playback systems.

4.3.2. The Random Access Video-disk System

The SMPTE controllable videodisk playback system developed by Russ Sasnett of the Media Lab's Film/Video section is a prime target for our virtual time code. Three requirements are cited for the development of "consumer (low-end) computer-aided video"—standardized software control of all video devices (i.e. RS422 remotes), addressing information stamped on all video materials (i.e. SMPTE time code), and an indexing hierarchy of the video contents. The goal is

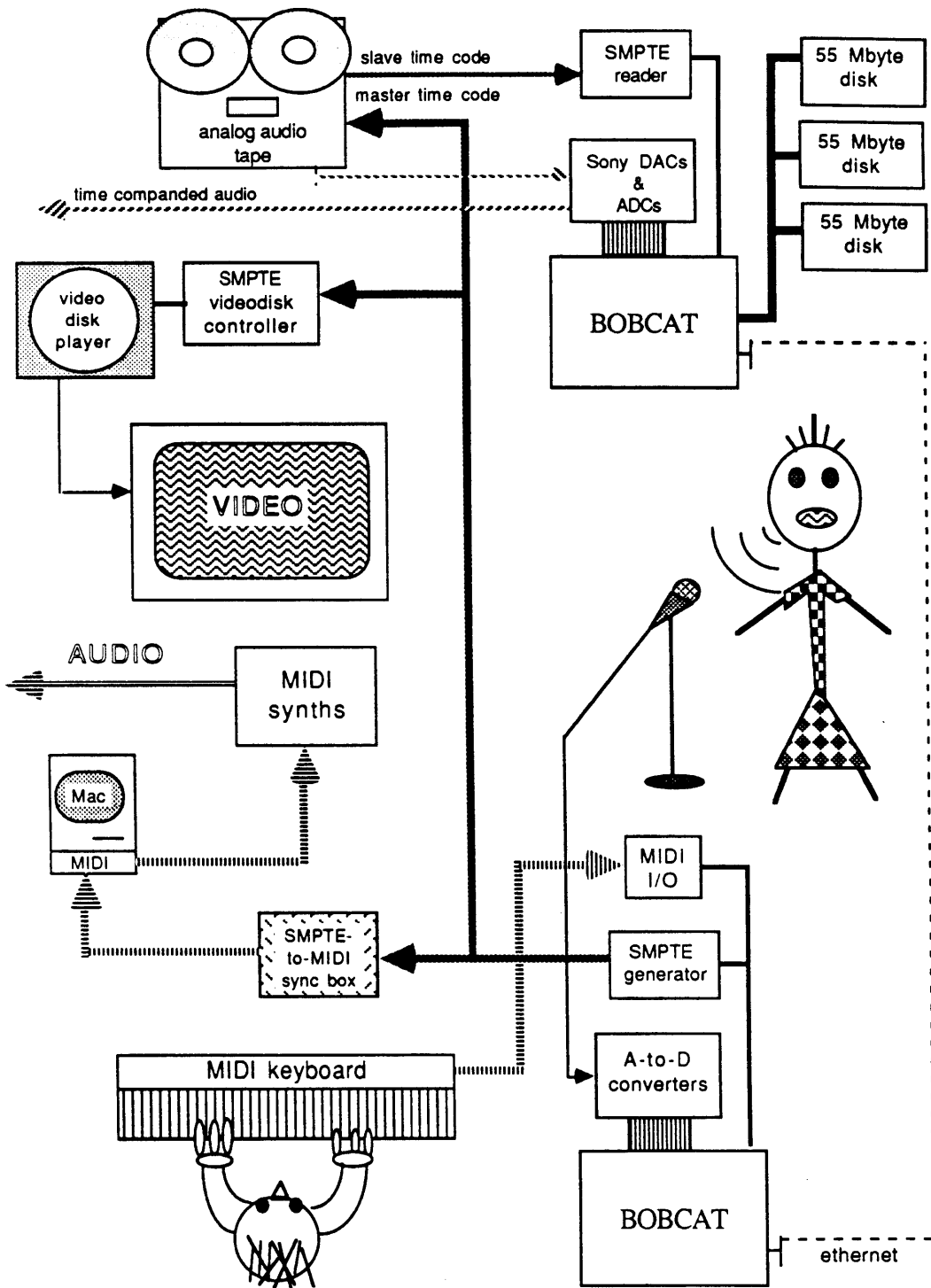


Figure 9: The Multi-media Performance System

for "video playback devices [to] become simply computer peripherals, with standardized methods for interfacing, storing and retrieving video segments, and building directory structures." [Sasnett, p. 15] Based on an IBM Personal Computer, the video disk system uses an RS232 equipped time code reader to provide incoming (master) SMPTE sync information to the PC, which in turn controls two Sony LDP-1000 video disk players. An RS422 controlled video switcher selects the output of whichever of the two players is producing the currently in-sync video. Typically, both players contain copies of the same video disk in order to produce minimal seek latency for true random access video. Video time compansion is achieved by skipping or repeating individual frames as necessary to make a video segment fit its allotted time. One of the newer "jump players"—such as the Phillips VP 935-ISI—would be capable of skipping 99 frames during the video vertical blanking interval, allowing much of the same variable-speed playback capability on a single video player. [Zvonar] Such a configuration allows a performance tracking system to find a virtual time point in a score or script of some sort, generate virtual time code for that point, and have the PC based video playback system correlate that point to its video index hierarchy and send the appropriate playback commands to the video disk players and switcher.

4.3.3. DSP Hardware for Real-time Systems: The Future is Soon

The future for the real-time performance system lies in personal computers with fast digital signal processing hardware. Since the host processor of a system such as the Bobcat workstation is kept quite busy with synchronization and coordination functions for its various peripherals, a signal processing card should be added to the system to assure real-time audio throughput. Similarly, a machine such as a MacIntosh II or even a MacIntosh SE would make a formidable music workstation if equipped with a state-of-the-art DSP card—such as one using the new

Motorola DSP56000 chip—and 16 bit sound, MIDI, and SMPTE I/O. Such a system begins to bring the cost of advanced studio capabilities into the realm of the small studio and the working musician.

5. Acknowledgements

Many thanks are due to the people of the Media Lab for their inspiration and help. Particular thanks go to Miller Puckette, Barry Vercoe, and John Amuedo for invaluable assistance with the signal processing end of things. Credit is also due to Richard Leacock, Russell Sasnett, Richard Zvonar, and others at MIT Film/Video for their original discussions of music and video performance synchronization.

Appendix A: Hardware

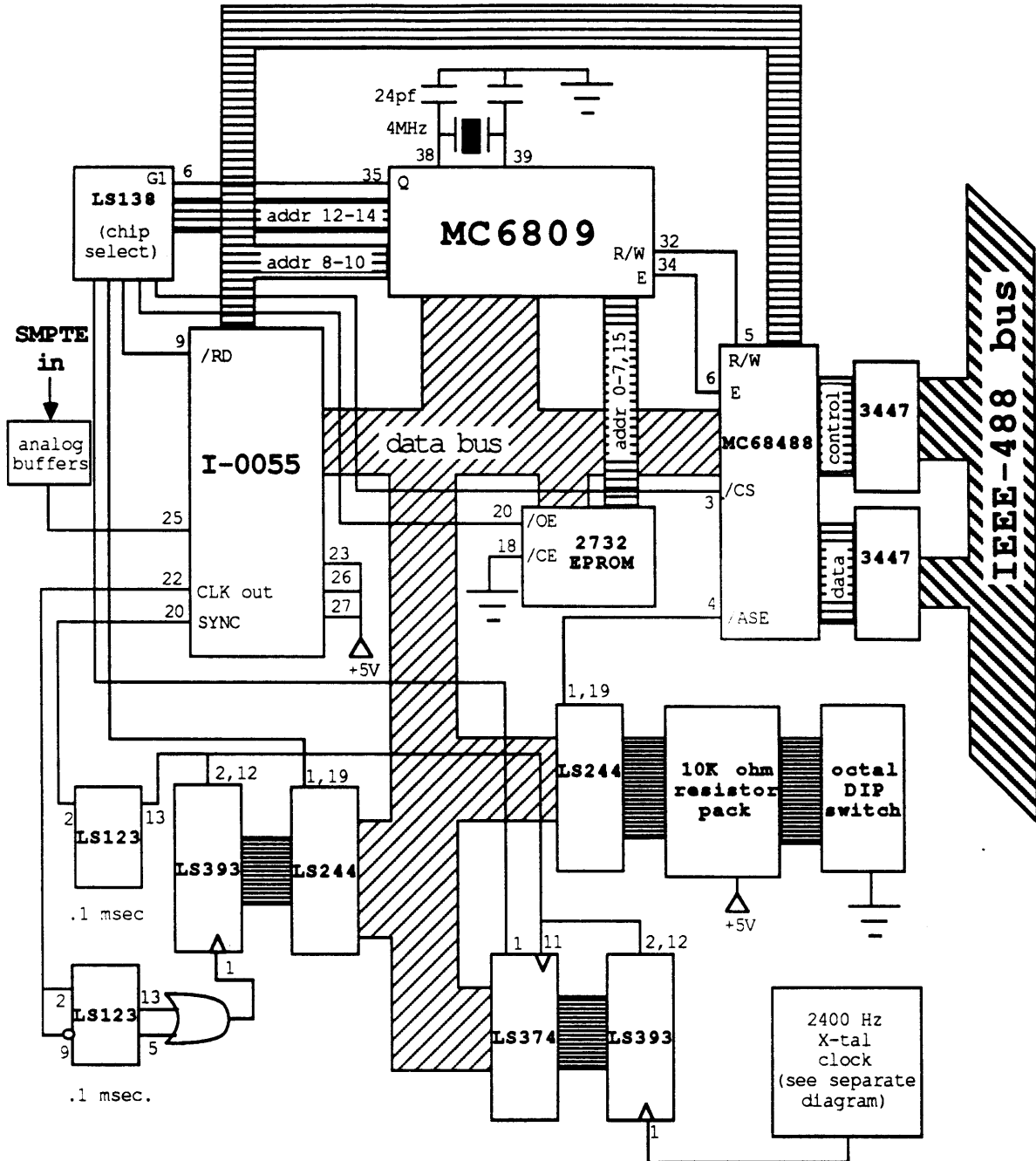


Figure A.1: SMPTE Time Code Reader Schematic

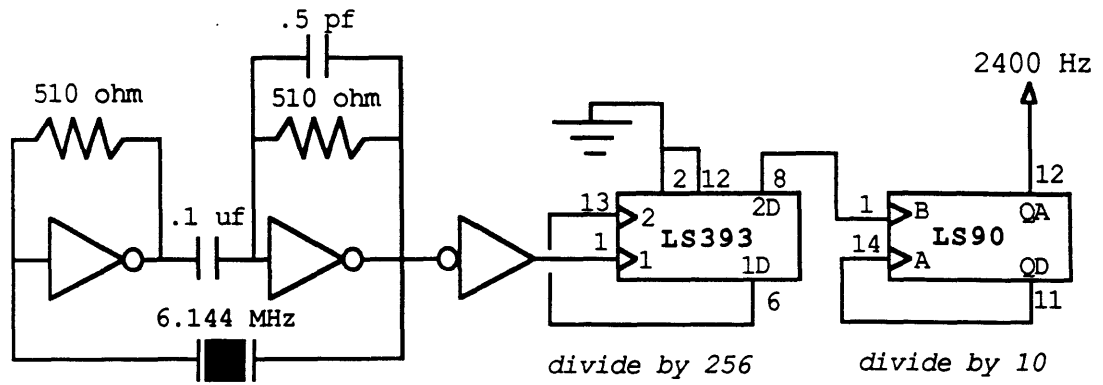


Figure A.2: 2400 Hz Crystal Clock for Time Code Reader

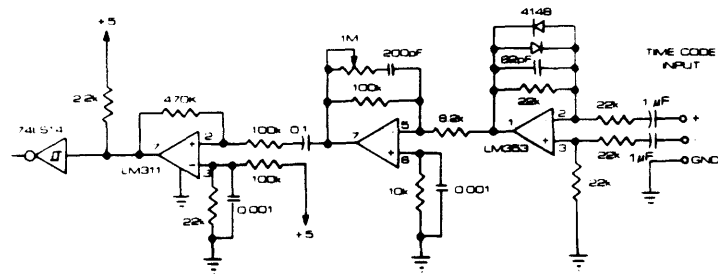


Figure A.3: Analog Time Code Buffers for Reader Card

MC6809 Address Table for Time Code Reader Card

\$00 00	EPROM program memory
\$00 FF	
	MC68488 registers (read/write):
\$10 XX	interrupt status/interrupt mask
\$11 XX	command status/unused
\$12 XX	address status/address mode
\$13 XX	auxiliary command/auxiliary command
\$14 XX	address switch/address
\$15 XX	serial poll/serial poll
\$16 XX	command pass-through/parallel poll
\$17 XX	data in/data out
	I-0055 registers:
\$20 XX	user bits #1
\$21 XX	user bits #2
\$22 XX	user bits #3
\$23 XX	user bits #4
\$24 XX	frames
\$25 XX	seconds
\$26 XX	minutes
\$27 XX	hours
\$3X XX	subframe register
\$4X XX	speed register
\$5X XX	unused
\$7X XX	
\$8X 00	BCD look-up tables in EPROM
\$8X FF	

DIP Switch Map for Time Code Reader Card

switch 0	ADDR0	primary device address for the
switch 1	ADDR1	time code reader on the IEEE-488
switch 2	ADDR2	bus (HP select code)
switch 3	ADDR3	.
switch 4	ADDR4	.
switch 5	MODE0	operating mode for card (see table below)
switch 6	MODE1	.
switch 7	MODE2	.

SMPTE Time Code Reader Operating Modes

mode 0	plain SMPTE
	returns [\$FF hours minutes seconds frames] as 5 bytes when polled
mode 1	plain SMPTE plus subframe
	returns [\$FF hours minutes seconds frames subframes] as 6 bytes when polled
mode 2	speed only
	returns [speed] as single byte when polled
mode 3	SMPTE plus subframe plus speed
	returns [\$FF hours minutes seconds frames subframes speed] as 7 bytes when polled
mode 4	user bits
	returns [\$FF user_bits_0 user_bits_1 user_bits_2 user_bits_3] as 5 bytes when polled
modes 5-7	software control
	user sends byte defined as follows
	bit 0 hours
	bit 1 minutes
	bit 2 seconds
	bit 3 frames
	bit 4 subframes
	bit 5 speed
	bit 6 user bits
	bit 7 unused
	and is returned
	[\$FF hours minutes seconds frames subframes speed user_bits]
	for n+1 bytes as requested (except for 4 user_bit bytes)

Appendix B: Software

6809 assembler code for Bobcat (IEEE-488) SMPTE reader

INIT:

```
LDA    $1400          read MC68488 address register
PSHU   $02            push value onto stack
ANDA   #$1F           mask user bits from address switch

STA    $1400          write address to MC68488
PULU   $02            restore unmasked value to A
ANDA   $E0            mask address bits

CMPA   #$00           check user bits for operating mode
BEQ    MODE0

CMPA   #$20           MODE1
BEQ    MODE1

CMPA   #$40           MODE2
BEQ    MODE2

CMPA   #$60           MODE3
BEQ    MODE3

CMPA   #$80           MODE4
BEQ    MODE4
```

more operating modes than you could shake a stick at

SOFTCONTROL:

```
LDA    $1000          read 68488 interrupt status register
LSRA                   shift right (into C)
BCC    SOFTCONTROL    loop if C is zero (wait for byte received)
SPIN:  LDB    $1300     read 68488 auxiliary command register
        ANDB   #$10     mask off bit 4 = RFD = ready for data
        CMPB   #$10     and test
        BNE    SPIN     spin if not RFD
        LDA    #$FF     load sync byte into accumulator
        STA    $1700

        LDA    $1700     read 68488 data in register
        LSLA                   shift left (into C)
        BCS    HOURSERVE
```

MINUTENEXT:

```
LSLA
BCS    MINUTESERVE
```

SECONDNEXT:

```
LSLA
BCS    SECONDSERVE
```

FRAMENEXT:

LSLA
BCS FRAMESERVE

SUBFRAMENEXT:

LSLA
BCS SUBFRAMESERVE

SPEEDNEXT:

LSLA
BCS SPEEDSERVE

UBNEXT:

LSLA
BCS UBSERVE

BRA SOFTCONTROL

servers for SOFTCONTROL mode

HOURSERVE:

JSR HOUR
BRA MINUTENEXT

MINUTESERVE:

JSR MINUTE
BRA SECONDNEXT

SECONDSERVE:

JSR SECOND
BRA FRAMENEXT

FRAMESERVE:

JSR FRAME
BRA SUBFRAMENEXT

SUBFRAMESERVE:

JSR SUBFRAME
BRA SPEEDNEXT

SPEEDSERVE:

JSR SPEED
BRA UBNEXT

UBSERVE:

JSR UB
BRA SOFTCONTROL

other modes

```
MODE0:
      JSR  VANILLA
      BRA  MODE0

MODE1:
      JSR  VANILLA
      JSR  SUBFRAME
      BRA  MODE1

MODE2:
      JSR  SPEED
      BRA  MODE2

MODE3:
      JSR  VANILLA
      JSR  SUBFRAME
      JSR  SPEED
      BRA  MODE3

MODE4:
      JSR  UB
      BRA  MODE4
```

* the real live subroutines that do all the dirty work

VANILLA:

```
SPIN:  LDB  $1300      read 68488 auxiliary command register
      ANDB #$10      mask off bit 4 = RFD = ready for data
      CMPB #$10      and test
      BNE  SPIN      spin if not RFD
      LDA  #$FF      load sync byte to accumulator
      STA  $1700     and write it to the MC68488 output register

      JSR  HOUR
      JSR  MINUTE
      JSR  SECOND
      JSR  FRAME
      RTS           return from subroutine
```

HOUR:

```
SPIN:  LDB  $1300      read 68488 auxiliary command register
      ANDB #$10      mask off bit 4 = RFD = ready for data
      CMPB #$10      and test
      BNE  SPIN      spin if not RFD
      LDX  $2700     read hours from I-0055 into X register
      LDA  [$8000,X] look-up binary for BCD value
      STA  $1700     and write it to 68488 out
      RTS           then return
```

MINUTE:

```
SPIN:  LDB  $1300      read 68488 auxiliary command register
      ANDB #$10      mask off bit 4 = RFD = ready for data
      CMPB #$10      and test
      BNE  SPIN      spin if not RFD
```

	LDX	\$2600	read minutes from I-0055 into X
	LDA	[\$8000,X]	look-up binary for BCD value
	STA	\$1700	and write it to 68488 out
	RTS		then return
SECOND:			
SPIN:	LDB	\$1300	read 68488 auxiliary command register
	ANDB	#\$10	mask off bit 4 = RFD = ready for data
	CMPB	#\$10	and test
	BNE	SPIN	spin if not RFD
	LDX	\$2500	read seconds from I-0055 into X
	LDA	[\$8000,X]	look-up binary for BCD value
	STA	\$1700	and write it to 68488 out
	RTS		then return
FRAME:			
SPIN:	LDB	\$1300	read 68488 auxiliary command register
	ANDB	#\$10	mask off bit 4 = RFD = ready for data
	CMPB	#\$10	and test
	BNE	SPIN	spin if not RFD
	LDX	\$2400	read frames from I-0055 into X
	LDA	[\$8000,X]	look-up binary for BCD value
	STA	\$1700	and write it to 68488 out
	RTS		then return
SUBFRAME:			
SPIN:	LDB	\$1300	read 68488 auxiliary command register
	ANDB	#\$10	mask off bit 4 = RFD = ready for data
	CMPB	#\$10	and test
	BNE	SPIN	spin if not RFD
	LDA	\$3000	read subframes counter
	STA	\$1700	and write it to 68488 out
	RTS		then return
SPEED:			
SPIN:	LDB	\$1300	read 68488 auxiliary command register
	ANDB	#\$10	mask off bit 4 = RFD = ready for data
	CMPB	#\$10	and test
	BNE	SPIN	spin if not RFD
	LDA	#\$50	load 80 decimal into accumulator
	SUBA	\$4000	and subtract the raw speed count
	STA	\$1700	and write it to 68488 out
	RTS		then return
UB:			
SPIN:	LDB	\$1300	read 68488 auxiliary command register
	ANDB	#\$10	mask off bit 4 = RFD = ready for data
	CMPB	#\$10	and test
	BNE	SPIN	spin if not RFD
	LDX	\$2000	read user bits 0 from I-0055 into X
	STA	\$1700	and write them to 68488 out
SPIN:	LDB	\$1300	read 68488 auxiliary command register
	ANDB	#\$10	mask off bit 4 = RFD = ready for data
	CMPB	#\$10	and test
	BNE	SPIN	spin if not RFD
	LDX	\$2100	read user bits 1 from I-0055 into X
	STA	\$1700	and write them to 68488 out
SPIN:	LDB	\$1300	read 68488 auxiliary command register
	ANDB	#\$10	mask off bit 4 = RFD = ready for data
	CMPB	#\$10	and test
	BNE	SPIN	spin if not RFD

	LDX	\$2200	read user bits 2 from I-0055 into X
	STA	\$1700	and write them to 68488 out
SPIN:	LDB	\$1300	read 68488 auxiliary command register
	ANDB	#\$10	mask off bit 4 = RFD = ready for data
	CMPB	#\$10	and test
	BNE	SPIN	spin if not RFD
	LDX	\$2300	read user bits 3 from I-0055 into X
	STA	\$1700	and write them to 68488 out
	RTS		then return


```
/*.....
a zero crossing looped pitch shifter
mbl 6-19-87
*****/

#include <errno.h>
#include <fcntl.h>
#include <math.h>

#define TBLSIZE 512
#define PI 3.1416

extern int errno;

main(argc,argv)
int argc;
char *argv[];
{
    long pointer;
    int infd, outfd, i, j, k, m, n;
    short sndbufi1[8192], sndbufi2[8192];
    short sndbufo[8192];
    unsigned char header[1024];
    double ratio, accumulator1, accumulator2, window[TBLSIZE], halfwind[16];

    if (argc != 4) {
        printf("usage: zeroc <input_file> <output_file> <1:n ratio>\n");
        exit(1);
    }

    if ((infd=open(argv[1],O_RDONLY)) < 0) {
        printf("error opening soundfile %s: errno = %d\n",argv[1],errno);
        exit(1);
    }

    if ((outfd=open(argv[2],O_WRONLY|O_CREAT|O_EXCL,0644)) < 0) {
        printf("error opening soundfile %s: errno = %d\n",argv[2],errno);
        exit(1);
    }

    for (i=0;i<TBLSIZE;i++) /* generate sin**2 table for large window */
        window[i] = pow(sin((double)(i*PI/TBLSIZE)),(double)2); /* DOUBLE !!! */

    for (i=0;i<16;i++) /* generate sin**2 table for short half-window */
        /* PI/2 is HALF a window */
        halfwind[i] = pow(sin((double)(i*PI/32)),(double)2);

    ratio = atof (argv[3]); /* ratio of shifted pitch to original */

    read (infd,header,1024);
    write (outfd,header,1024);

    accumulator1=accumulator2=0;

    n = sizeof(sndbufi2);
    m = read (infd,sndbufi1,sizeof(sndbufi1));
    pointer = 1024L;
    i = 0;

    while (m==sizeof(sndbufi1)&& n==sizeof(sndbufi2)) {
```

```
/* read unscaled table values for 512 input samples, then
   look for a simple negative direction zero crossing */

pointer += 2*i;

for (i=0;(accumulator1<512)||
     (sndbufi1[(int)accumulator1+1]>0)||(sndbufi1[(int)accumulator1]<=0)
     ;i++) {
/*   insert ratio change for real-time control here   */
   sndbufo[i] = sndbufi1[(int)accumulator1];
   accumulator1+=ratio;
}

lseek (infd,pointer+(2*i),0);
n = read (infd,sndbufi2,sizeof(sndbufi2));

for (j=1;(sndbufi2[j]>0)||(sndbufi2[j-1]<=0);j++);
accumulator2=j;
for (k=0;k<16;i++,k++) {
   sndbufo[i]=(halfwind[15-k]*sndbufi1[(int)accumulator1]) +
   (halfwind[k]*sndbufi2[(int)accumulator2]);
/*   insert ratio change for real-time control here   */
   accumulator1+=ratio;
   accumulator2+=ratio;
}

for (;(accumulator2<512)||
     (sndbufi2[(int)accumulator2+1]>0)||(sndbufi2[(int)accumulator2]<=0)
     ;i++) {
   sndbufo[i]=sndbufi2[(int)accumulator2];
/*   insert ratio change for real-time control here   */
   accumulator2+=ratio;
}

lseek (infd,pointer+(2*i),0);
m = read (infd,sndbufi1,sizeof(sndbufi1));

for (j=1;(sndbufi1[j]>0)||(sndbufi1[j-1]<=0);j++);
accumulator1=j;
for (k=0;k<16;i++,k++) {
   sndbufo[i]=(halfwind[15-k]*sndbufi2[(int)accumulator2]) +
   (halfwind[k]*sndbufi1[(int)accumulator1]);
/*   insert ratio change for real-time control here   */
   accumulator1+=ratio;
   accumulator2+=ratio;
}

write (outfd,sndbufo,2*i);

}

}
```

```
/*
*****
          a pitch shifter with 50% overlapping windows
          mbl 7-9-87
*****
#include <errno.h>
#include <fcntl.h>
#include <math.h>

#define TBLSIZE 2048
#define PI 3.1416

extern int errno;

main(argc,argv)
int argc;
char *argv[];
{
    long pointer;
    int  infd, outfd, i, j, k, m, n, tblsizefour;
    short sndbufi1[TBLSIZE*4], sndbufi2[TBLSIZE*4];
    short sndbufo[TBLSIZE*4];
    unsigned char header[1024];
    double ratio, accumulator1, accumulator2, window[TBLSIZE];

    if (argc != 4) {
        printf("usage: wind <input_file> <output_file> <1:n ratio>\n");
        exit(1);
    }

    if ((infd=open(argv[1],O_RDONLY)) < 0) {
        printf("error opening soundfile %s: errno = %d\n",argv[1],errno);
        exit(1);
    }

    if ((outfd=open(argv[2],O_WRONLY|O_CREAT|O_EXCL,0644)) < 0) {
        printf("error opening soundfile %s: errno = %d\n",argv[2],errno);
        exit(1);
    }

    for (i=0;i<TBLSIZE/2;i++) /* generate sin**2 table for large window */
        window[i] = pow(sin((double)(i*PI/(TBLSIZE/2))),(double)2);
    for (;i<TBLSIZE;i++) /* and zero guard points */
        window[i] = 0.0;

    ratio = atof (argv[3]); /* ratio of shifted pitch to original */

    read (infd,header,1024);
    write (outfd,header,1024);

    tblsizefour=TBLSIZE/4;

    accumulator1=0;
    accumulator2=tblsizefour-1;

    m = read (infd,sndbufi1,sizeof(sndbufi1));
    n = sizeof(sndbufi2);
    pointer = 1024L;
    i = 0;

    while (m==sizeof(sndbufi1)&& n==sizeof(sndbufi2)) {
/* scale first 256 input samples by first half of sine table,
   then scale another 256 samples by second half of sine table */

```

```
pointer += 2*i;

for (i=0;(accumulator1<tblsizofour);i++) {
    sndbufo[i] = sndbufi1[(int)accumulator1]*
        window[(int)accumulator1]+
        sndbufi2[(int)accumulator2]*
        window[tblsizofour-1+(int)accumulator1];
    accumulator1+=ratio;
    accumulator2+=ratio;
}

lseek (infd,pointer+2*i,0);
n = read (infd,sndbufi2,sizeof(sndbufi2));
accumulator2=0;

for (;(accumulator2<tblsizofour);i++) {
    sndbufo[i]=sndbufi2[(int)accumulator2]*
        window[(int)accumulator2]+
        sndbufi1[(int)accumulator1]*
        window[tblsizofour-1+(int)accumulator2];
    accumulator1+=ratio;
    accumulator2+=ratio;
}

lseek (infd,pointer+2*i,0);
m = read (infd,sndbufi1,sizeof(sndbufi1));
accumulator1=0;

write (outfd,sndbufo,2*i);
}
}
```

```
/*
*****
          a pitch shifter looped at zero crossings
          with 50% overlapping windows
          using large RAM buffers instead of multiple reads and writes.
          mbl 7-6-87
*****
*/

#include <errno.h>
#include <fcntl.h>
#include <math.h>

#define BUFSIZE 1048576
#define TBLSIZE 4000
#define PI 3.1416

extern int errno;

main(argc,argv)
int argc;
char *argv[];
{
    long pointer;
    int  infd, outfd, i, j, k, m, n, offset1, offset2, tblsofour=TBLSIZE/4;
    short *malloc();
    short *sndbufi=malloc(BUFSIZE);
    short *sndbufo=malloc(BUFSIZE);
    unsigned char header[1024];
    double ratio, accumulator1, accumulator2, window[TBLSIZE];

    if (argc != 4) {
        printf("usage: newzw <input_file> <output_file> <1:n ratio>\n");
        exit(1);
    }

    if ((infd=open(argv[1],O_RDONLY)) < 0) {
        printf("error opening soundfile %s: errno = %d\n",argv[1],errno);
        exit(1);
    }

    if ((outfd=open(argv[2],O_WRONLY|O_CREAT|O_EXCL,0644)) < 0) {
        printf("error opening soundfile %s: errno = %d\n",argv[2],errno);
        exit(1);
    }

    for (i=0;i<TBLSIZE/2;i++) /* generate sin**2 table for large window */
        window[i] = pow(sin((double)(i*PI/(TBLSIZE/2))),(double)2);
    for (;i<TBLSIZE;i++) /* and zero guard points */
        window[i] = 0.0;

    ratio = atof (argv[3]); /* ratio of shifted pitch to original */

    read (infd,header,1024);
    write (outfd,header,1024);

    accumulator1=0;
    accumulator2=tblsofour-1;

    if ((m = read (infd,sndbufi,BUFSIZE))<0)
        printf ("error reading soundfile into buffer: errno = %d\n",errno);
    pointer = 1024L;
    offset1 = offset2 = 0;
    i = 0;

    while (m/2>i-tblsofour*2) { /* add some overwrite margin */

```

```
/* scale first 256 input samples by first half of sine table,
then look for a simple negative direction zero crossing,
then scale another 256 samples by second half of sine table
while fading in overlapping window */

for (;(accumulator1<tblsofour);i++) {
    sndbufo[i] = sndbufi[offset1+(int)accumulator1]*
        window[(int)accumulator1]+
        sndbufi[offset2+(int)accumulator2]*
        window[tblsofour-1+(int)accumulator1];
    accumulator1+=ratio;
    accumulator2+=ratio;
}

for (;(sndbufi[offset1+(int)accumulator1+1]>0)||
(sndbufi[offset1+(int)accumulator1]<=0);
i++) {
    sndbufo[i] = sndbufi[offset1+(int)accumulator1];
    accumulator1+=ratio;
}

offset2+=tblsofour;
for (;(sndbufi[++offset2]>0)|| (sndbufi[offset2-1]<=0););
accumulator2=0;

for (;(accumulator2<tblsofour);i++) {
    sndbufo[i]=sndbufi[offset2+(int)accumulator2]*
        window[(int)accumulator2]+
        sndbufi[offset1+(int)accumulator1]*
        window[tblsofour-1+(int)accumulator2];
    accumulator1+=ratio;
    accumulator2+=ratio;
}

for (;(sndbufi[offset2+(int)accumulator2+1]>0)||
(sndbufi[offset2+(int)accumulator2]<=0);
i++) {
    sndbufo[i] = sndbufi[offset2+(int)accumulator2];
    accumulator2+=ratio;
}

offset1+=tblsofour;
for (;(sndbufi[++offset1]>0)|| (sndbufi[offset1-1]<=0););
accumulator1 = 0;
}

write (outfd,sndbufo,(offset1>offset2?offset1*2:offset2*2));
}
```

```
/*
*****
a pitch shifter using the msp/bv constant difference pitch tracking
technique to determine period and window size.
this version uses 50% overlapping sine-squared windows.
mbl 8-7-87
*****
*/

#include <errno.h>
#include <fcntl.h>
#include <math.h>

#define BUFSIZE 1048576
#define TBLSIZE 2048
#define PI 3.1416

extern int errno;

long lambda_was(), lambda_peak();
long lambda, minlambda, maxlambda, minlam, maxlam;
short *hipeaks[200], *lopeaks[200];

main(argc,argv)
int argc;
char *argv[];
{
    long pointer;
    int infd, outfd, i, j, k, m, n, offset1, offset2, tblsofour=TBLSIZE/4;
    short *malloc();
    short *sndbufi=malloc(BUFSIZE);
    short *sndbufo=malloc(BUFSIZE);
    unsigned char header[1024];
    double ratio, accumulator1, accumulator2, window[TBLSIZE];

    if (argc != 4) {
        printf("usage: lambshift <input_file> <output_file> <1:n ratio>\n");
        exit(1);
    }

    if ((infd=open(argv[1],O_RDONLY)) < 0) {
        printf("error opening soundfile %s: errno = %d\n",argv[1],errno);
        exit(1);
    }

    if ((outfd=open(argv[2],O_WRONLY|O_CREAT|O_EXCL,0644)) < 0) {
        printf("error opening soundfile %s: errno = %d\n",argv[2],errno);
        exit(1);
    }

    for (i=0;i<TBLSIZE/2;i++) /* generate sin**2 table for large window */
        window[i] = pow(sin((double)(i-PI/(TBLSIZE/2))), (double)2);
    for (;i<TBLSIZE;i++) /* and zero guard points */
        window[i] = 0.0;

    ratio = atof (argv[3]); /* ratio of shifted pitch to original */

    read (infd,header,1024);
    write (outfd,header,1024);

    minlambda = 10;
    maxlambda = tblsofour;

    accumulator1=0;
    accumulator2=tblsofour-1;
}
```

```
if ((m = read (infd,sndbufi,BUFSIZE))<0)
    printf ("error reading soundfile into buffer: errno = %d\n",errno);
pointer = 1024L;
offset1 = offset2 = 0;
i = 0;

while (m/2>i-tblsofour*2) { /* add some overwrite margin */

/* scale first 512 input samples by first half of sine table,
then call lambda_was to determine a period on which to loop,
then scale another 512 samples by second half of sine table
while fading in overlapping window at the calculated point */

    for (;(accumulator1<tblsofour);i++) {
        sndbufo[i] = sndbufi[offset1+(int)accumulator1]*
            window[(int)accumulator1]+
            sndbufi[offset2+(int)accumulator2]*
            window[tblsofour-1+(int)accumulator1];
        accumulator1+=ratio;
        accumulator2+=ratio;
    }

    offset2+=tblsofour;
    lambda = lambda_was(&sndbufi[offset2],tblsofour);
    printf ("lambda at point %d is %d \n",offset2,(int)lambda);
    if (lambda != 0) {
        offset2 = offset1 + (int)accumulator1 - (2*lambda);
    }
    accumulator2=0;

    for (;(accumulator2<tblsofour);i++) {
        sndbufo[i]=sndbufi[offset2+(int)accumulator2]*
            window[(int)accumulator2]+
            sndbufi[offset1+(int)accumulator1]*
            window[tblsofour-1+(int)accumulator2];
        accumulator1+=ratio;
        accumulator2+=ratio;
    }

    offset1+=tblsofour;
    lambda = lambda_was(&sndbufi[offset1],tblsofour);
    printf ("lambda at point %d is %d \n",offset1,(int)lambda);
    if (lambda != 0) {
        offset1 = offset2 + (int)accumulator2 - (2*lambda);
    }
    accumulator1=0;
}

write (outfd,sndbufo,(offset1>offset2?offset1*2:offset2*2));
}
```


Bibliography

Bloch, Joshua, and Roger Dannenberg, "Real-Time Computer Accompaniment of Keyboard Performances," *Proceedings of the 1985 International Computer Music Conference*, The Computer Music Association, San Francisco.

Computer Music Journal, MIT Press, Cambridge. Periodical—articles on digital music processing and performance interfaces.

Dannenberg, Roger, "An On-Line Algorithm for Real-Time Accompaniment," *Proceedings of the 1984 International Computer Music Conference*, The Computer Music Association, San Francisco.

Dolson, Mark, "The Phase Vocoder: A Tutorial," *Computer Music Journal*, Cambridge, 1986. Vol. 10, No. 4, pp. 14-27. High fidelity audio analysis and resynthesis with specific applications for time compansion and pitch shifting.

EECO Incorporated, *SMPTE/EBU Longitudinal and Vertical Interval Time Code*, Santa Ana, CA, 1982. Time code formats and applications.

Factor, Richard, "A 6.4 Second Delay Line for Broadcast Applications", *Preprints: AES 61st Convention, New York City, 1978 November 3-6*, Audio Engineering Society, New York, 1978. Preprint 1417 (I-7). Audio time expansion for live broadcast applications.

Journal of the Audio Engineering Society. Periodical—articles on audio signal processing and synchronization.

Leacock, Richard, "Speculations on what we need to do to make an interactive opera possible," unpublished manuscript, MIT Media Laboratory, Cambridge, MA, March 1987.

Lee, Francis, "Time Compression and Expansion of Speech by the Sampling Method," *Journal of the Audio Engineering Society*, New York, 1972. Vol. 20, No. 9, pp. 738-742. Early paper on digital means of audio time compansion.

LoCascio, Marc, "Digital-Analog Convertors for Low-Cost Computer Music Workstations," bachelor's thesis, MIT Media Laboratory, Cambridge, MA, February 1986.

Mellado, Ramon, "MIDI to UNIX Workstation Interface," bachelor's thesis, MIT Media Laboratory, Cambridge, MA, May 1987.

Puckette, Miller, "Interprocess Communication and Timing in Real-time Computer Music Performance," *Proceedings of the 1986 International Computer Music Conference*, The Computer Music Association, San Francisco.

Puckette, Miller, "A Score Following Algorithm," unpublished manuscript.

Runstein, Robert E., and David Miles Huber, *Modern Recording Techniques*, Howard W. Sams & Co., Indianapolis, 1986. Time code format, synchronization techniques, professional recording studio practices.

Sasnett, Russell, "Reconfigurable Video," master's thesis, MIT Media Laboratory, Cambridge, MA, February 1986.

Schmidt, Brian, and James Roth, "The Synchronization of Audio Production in Computer Music," *Proceedings of the 1985 International Computer Music Conference*, The Computer Music Association, San Francisco.

Vercoe, Barry, "The Synthetic Performer in the Context of Live Performance", *Proceedings of the 1984 International Computer Music Conference*, The Computer Music Association, San Francisco.

Woram, John M., *The Recording Studio Handbook*, ELAR Publishing Company, Plainview, NY, 1982. Time code format, synchronization techniques, audio signal processing, professional recording studio practices.

Zvonar, Richard, "Peace Project—Performance Tracking System," unpublished manuscript, MIT Media Laboratory, Cambridge, MA, March 1986.