Development of An Improved Swept RF Tagging System and its Musical Applications

By

Laurel P. Smith

B.S., Electrical Engineering and Computer Science (2001)

Massachusetts Institute of Technology

Submitted to the Department of Electrical Engineering and Computer Science
In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Computer Science and Electrical Engineering

At the

Massachusetts Institute of Technology

January 2002

Signature of Author . . . . . .
Department of Electrical Engineering and Computer Science
January 1, 2002

Certified by . . . . . .
Joseph A. Paradiso
Principal Research Scientist, Media Arts and Sciences
Thesis Supervisor

Accepted by . . . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

Development of An Improved Swept RF Tagging System and its Musical Applications

By

Laurel P. Smith

Submitted to the Department of Electrical Engineering and Computer Science
On February 6, 2001 in partial fulfillment of the requirements for the
Degree of Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

Building on the previous work and an earlier design by Dr. Joe Paradiso and Kai-Yuh Hsiao, an improved swept frequency tag reader has been developed and built. The tag reader uses an AC magnetic field to detect the presence and coupling strength of magnetically-coupled resonators, in this case, passive resonant LC circuits and magnetostrictors. The previously designed swept RF tag reader is capable of simultaneously detecting the continuous motion of up to 20 distinct tags in real-time within roughly 12" distance from a 12'' diameter search coil. The new reader is capable of matching that performance, but also features increased sensitivity, vastly improved tolerance to frequency drift and support for up to 3 coils to be run asynchronously, a useful feature for multi-coil geometries that enable multi-axis tracking and response

Using a single coil, a new musical application, Musical-Navigatrics has been developed for the tag reader. This application works to provide an expressive and complex free space musical interaction for composition and performance specifically suited to the tagged object interface. Moving tags above the search coil, the user is able to play and record up to 4 pitched voices and a percussive line. These voices can then be distorted and explored through the use of a variety of effect inducing tags. Through this application, the capabilities and the usability of the tag interface has been explored for usage as a pitched musical instrument, effects controller, and basic sequencer, features that can create an entirely new and revolutionary performance controller for computer music.

Thesis Supervisor: Joseph A. Paradiso

Title: Principal Research Scientist, Media Arts and Sciences

Acknowledgements

First and foremost I would like to thank Joe Paradiso for giving me the opportunity to work on this project as well as his help, knowledge, faith, and support throughout, especially considering my constrained military schedule, Leila Hasan for being the best office mate I could ever ask for and so much more, Matt Hancher for always answering my questions, Mark Feldmeier for his analog guidance and writing such interesting e-mails, Candyland for giving me a home when I had none, Vis Taraz and Marc Rios for always causing the necessary ruckus, Romy Shioda for your incredible musicianship and playing crazy music with me for my five years here, the MIT Music Department, the Herbs, and everyone else I have played with for being such inspirations and the reason my MIT experience was the pleasure it was, and finally, the United States Air Force for putting me through college, making this entire experience possible and sending me to Turkey.

Table of Contents

Table of Figures

Chapter 1

Introduction

Recent history has seen tremendous expansions in the capability and power of computational machines in a short amount of time. In merely 50 years, we have seen the introduction of computers and their evolution from being giant, expensive, and exclusive university research platforms to becoming one of the most ubiquitous tools today. Computers are now widely available everywhere that technology reaches and are used for a vast array of activities. However, despite this growth in both computational power and computers' popularity, the computer interface has remained largely stagnant. The present day graphical user interface is tied to 2-D tracking and display technology, with the keyboard and mouse as the primary mode of interaction. New investigations into computer human interfaces to provide more natural and sensible interactions are looking to move beyond these restrictive boundaries.

Not unlike computers, in the past 20 years, the popularity of entirely electronic music has skyrocketed. As electronic music takes advantage of the new ubiquity of processing power, the wide capabilities for sound manipulation have made it an attractive creative platform while the drop in processor cost has made it possible to purchase powerful music computation engines for an accessible price. The number of people using electronic synthesizers and production tools has dramatically risen, and along with that, electronic music performances have become common well-attended events. Yet despite this outburst in the creativity of new sounds, styles, and tools, electronic music interfaces have remained largely the same: buttons, knobs, keyboards and the standard computer GUI. What these all lack is an expressive interface capable of manipulating the

full scope of what processors have made musically available. Most electronic musicians, unlike acoustic musicians, remain largely unable to interact with the rich sonic soundscape of this new music at an expressive and gestural level that is reciprocal to the expressivity, complexity, and form of the music itself. In creation and performance they remain tied to small interfaces that allow for only minimal physical interaction. Although an expressive physical interface is not a requirement for interesting music, the physical interactions between an instrument and a performer are key to a rich rewarding experience, as any acoustic player has experienced.

Worse is live performance. While such performances are increasingly popular, musicians continue to have difficulty finding means to translate their music from the dry interaction of the production studio to an interesting, entertaining, and rousing live show. Some musicians resort to transcribing the music for live instruments, putting dancers on the stage, or using graphical imagery to distract from the lack of on-stage activity. A worst-case performance, which is not by any means uncommon, consists of the performers standing in front of computers and interacting through the standard GUI. Not surprisingly, this does not generate much viewer engagement.

Recognizing the need for new interactive interfaces, the Responsive Environments Group, directed by Dr. Joe Paradiso at the MIT Media Lab, has designed and built a swept-frequency passive tag reader[1]. This device is capable of continuously sensing and identifying at least 20 distinct small passive tags within a limited range in real-time. The basic operation is achieved by driving a search coil at various frequencies to create an AC magnetic field in the active region and then introducing small magnetically-coupled resonators (e.g. LC circuit tags or magnetostrictiors) into this field. If the field is driven at a tag's resonant frequency, the tag will draw energy from the magnetic field, altering the current through the search coil. This change in coil inductance can be detected and matched with frequency, allowing identification of the resonant tag. The magnitude

of change corresponds to the amount of coupling, which is determined by the tag's distance and orientation to the field. As only one tag will couple with the coil at a specific frequency, sweeping through a range of frequencies allows simultaneous detection of tags; these sweeps can be performed fast enough to provide real-time interaction. Due to their small size, these resonant tags can be embedded into a wide range of objects, creating new tangible interfaces[2] and, consequently, new musical controllers.

1.1 Introduction to Magnetic Tags as Sensors

Magnetic tagging systems are already a well-established technology. A wide variety and range of applications already exist in which tags are commonly used. Tagging system are available to perform complex tasks, as in the case of active tagging systems that have been developed for accurate motion capture, to simple ones, such as the “Electronic Article Surveillance” (EAS) tagging systems often used to detect shoplifting.

Passive tags, as opposed to battery powered or wired active tags that often require potentially complicated hardware, are tags that draw power transmitted by an antenna to trigger some kind of detectable response. They generally fall into two categories, chip-based radio frequency identification, RFID[3] tags or chip-less resonant tags. RFID tags work by powering a CMOS state-machine chip, which then transmits some sort of unique identifying response. Removing the micro-controller essentially results in a magnetically-coupled resonance tag, a tag whose identification is made only by the resonant frequency at which it draws power.

In order to work as a tangible or musical interface, the tagging system must be able to detect a variety of tags in real-time over a reasonable distance. RFID tags, due to the time required for them to draw the power necessary for the chip and provide an anti-collision response when more than one tag is present, generally respond too slowly to work in a real-time system; the remaining

option is magnetically-coupled resonant tags. The primary drawback of magnetically-coupled resonant tags is the limited number of identities available, as the tag's response is not precise. The relation of a tag's coupling strength to the width of the response in a frequency sweep is described as the tag's *Q*. Using higher *Q* tags allows for more tags to fit within the same frequency sweep.

Although resonant tags have been limited to predominantly EAS systems, other devices have explored using passive, magnetically-resonant tags for detection and tracking purposes. The classic *Wacom Tablet*[4] uses resonant tags to track and identify coded whiteboard pens, while one of its descendants in the musical controller world, Don Buchla's *Marimba Lumina*[5], embeds tags into the marimba mallets enabling them to act as MIDI controllers. Both these interactions however, are very close-range, acting within only an inch or so above the interactive surface.

Chapter 2

Passive Tag Readers

Development of passive resonant tags to explore new possibilities for computer-human interfaces was begun by Dr. Joseph Paradiso and Kai-Yuh Hsiao within the Responsive Environments research group at the MIT Media Lab. In order to serve as a tangible interface, the primary two requirements for a successful tag reader are that it has to be able to detect and track a substantial number of tags, and that it has to do it in real-time. Along with that, the reader should be able to detect a continuous range of coupling amplitudes and needs to be relatively stable. There have been two approaches to passive tag reading explored throughout the course of their research: pulsed ringdown devices and continuous swept-frequency architectures.

2.1 Implementation of a Ringdown Tag Reader

A first attempt[1] at implementing a passive tag interface was done by exploiting "ringdown" to identifying multiple tags. The operational basis for a ringdown reader derives from transmitting a magnetic pulse at the resonant frequency of a tag. The tag is effectively energized by the pulse and releases this energy as an electronic ringdown response when the driving magnetic pulse stops. This resonant response is small, but can be clearly detected by a sensitive listening reader.

Using this principle, a ringdown tag reader was built that would "ping" or transmit at a specific discrete known frequency and then switch off, allowing read circuitry to listen for and detect the ringdown that would occur if a tag was present at that transmit frequency. In order to detect multiple tags, the reader cycled through, "pinging" at the resonant frequencies of multiple

tags. To better match the read coil to the desired driving resonance, a ladder of capacitors and triacs was added to enable dynamic retuning.

With this strategy, the ringdown tag reader was able to successfully detect multiple tags continuously and could complete a full ping-listen cycle 30 times a second. The system suffered, however, as the detection process for one tag took roughly 12 milliseconds. Thus, to remain operating in real-time, the number of tags is very limited. Assuming a minimum 10 Hz update rate, only 8 tags could be used. Also, the system was particularly sensitive to the frequency drift inherent in the drive system and high-*Q* tags, as transmission needed to occur at specific, discrete, well-tuned spots. Although the ringdown reader worked well, these drawbacks prevented it from fully meeting the needs of the desired tangible interface.

2.2 Swept Frequency Tagging

Recognizing the restrictions placed on the ringdown system, it was decided to move from discrete frequency steps to a continuous sweep. As there is now continuous "pinging," a swept-frequency reader required a change in its detection mechanism, using a continuously operational inductive bridge instead of the discretely switched listening circuitry used with the ringdown approach. The swept frequency tag reader that was built[1] sweeps from roughly 400kHz down to around 50 kHz. The sweep frequency curve is shaped exponentially, as this helps balance the tag coupling between the high and low frequencies. This problem stemmed from the fact that a linear sweep will drive far more wave cycles of a high frequency than a low frequency given the same amount of time. Using a linear sweep resulted in the higher frequencies being driven more strongly, an effect countered by reshaping the sweep to spend more time at lower frequencies so that the time spent at a frequency is indirectly related to the frequency itself. The goal is to balance

the received signal from the tags at all frequencies so they are all optimized for maximum response in minimum time. Lastly, while the ringdown reader preferred magnetostrictors (a common materials-based EAS tag) due to their high *Q* and strong ringdown response, the swept-frequency reader was better suited for LC (coupled inductor-capacitor) tags. Although the signal produced by most LC tags is usually not as strong as for magnetostrictors, the ringdown, which can interfere with subsequent tags encountered later in the sweep, is reduced, and the tuning of LC tags is fairly easy, clean, stable, and reliable.

As indicated, tag detection is achieved using an inductive bridge (see Figure 2-1). As the tag perturbs the magnetic field, it alters the current running through the search coil. Four inductors, one of which is the sense, or search, coil, form the inductive bridge. The non-search-coil leg of the bridge serves as a tunable reference that does not interact with the resonant tags. This enables the comparison of the current through the two legs, leading to the determination of the differential voltage and thus signal presence and strength. This signal is further processed to transform it into a well-sized clean DC signal as shown in Figure 2-1. The original tag reader uses a PIC16C73 and

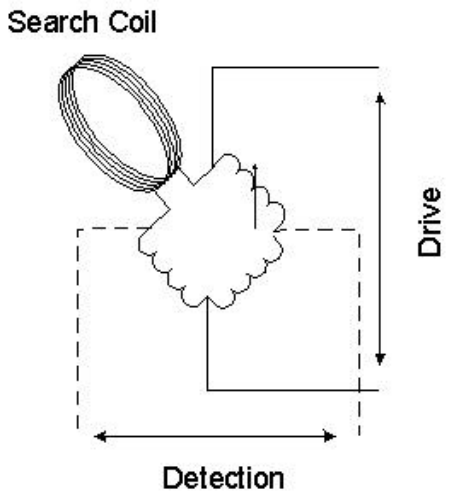


Figure 2-1: Bridge Circuit

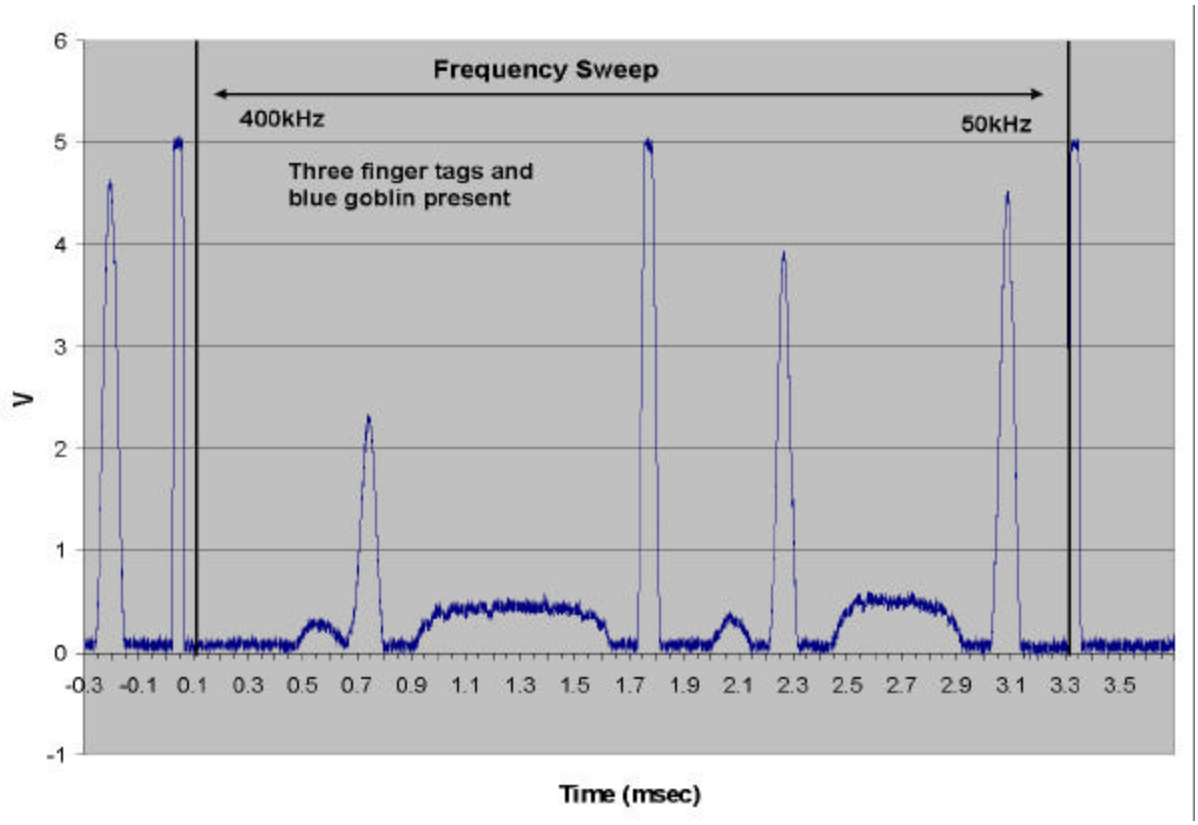


Figure 2-2 *Analog* Output for Swept Frequency Tagging (with multiple tags present)

presence and strength. This signal is further processed using analog gain and filter stages that transform it into a well-sized clean DC signal as shown in Figure 2-2 The original tag reader uses a PIC16C73 and later a PIC16C83 micro-controller to interpret this signal strength and send information on tag frequency and strength to a host PC through a serial link.

This swept frequency tag reader achieved the primary tangible interface goals fairly well. The reader completes a full sweep of the spectrum 30 times a second, correctly sensing and identifying tags in real-time. The number of tags that can be sensed is now, unlike the ringdown reader, limited primarily by frequency overlap between tag responses. Due to parasitic capacitances, the frequency at which the tag will respond is not entirely specific, and the tag's *Q*, determines the tightness of the response. Depending on their components, LC tags have varying *Q*

thus the resonant frequencies need to be appropriately spread out to minimize interaction between tags. Despite this problem, the swept-tag reader, sweeping across roughly a decade in frequency, is still able to easily detect as many as 20 tags at a time. This number could be expanded considerably by increasing the range of the frequency sweep, and by more careful selection and creation of tags. Without slowing the real-time refresh rate, increasing the sweep range will reduce the time spent at any given tag resonance, weakening the response but allowing for more tags. Detection is also continuous, and at present has a reasonable range of roughly 12" from the search coil, depending on the build and frequency of the tag.

Chapter 3

Improvements to the Swept RF Tag Reader

While the first implementation of the tag reader by Dr. Paradiso and Hsiao was certainly successful, there remained a number of improvements that could be made to its final design[1]. As part of this thesis, a new tag reader, based on the previous version, has been built in order to rectify some of the identified problems, as well as enable new means for improved tag reading capability. These improvements are aimed at increasing the tag reader's sensitivity and dynamic range, stability, and expandability. The circuit can be broken down into three functional areas: frequency sweep generation, inductive sensing, and system control. All three of these areas have been modified in the course of this thesis. The resulting circuit board is shown in Figure 3-1 and a block diagram of the circuit as shown in Figure 3-2.

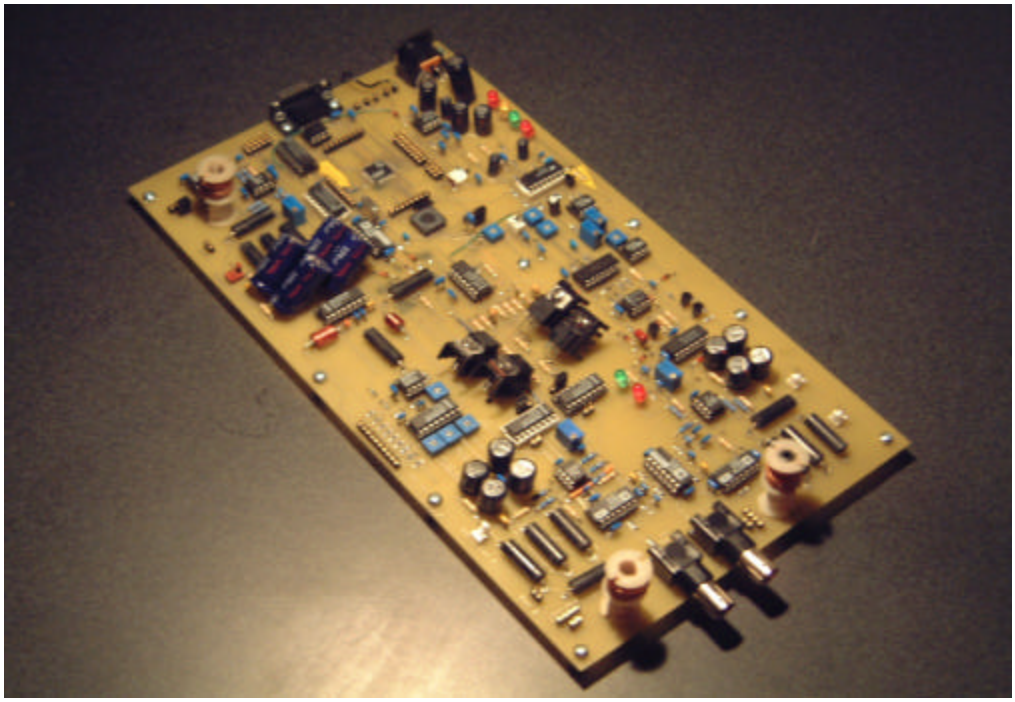


Figure 3-1: Most Recent Tag Reader

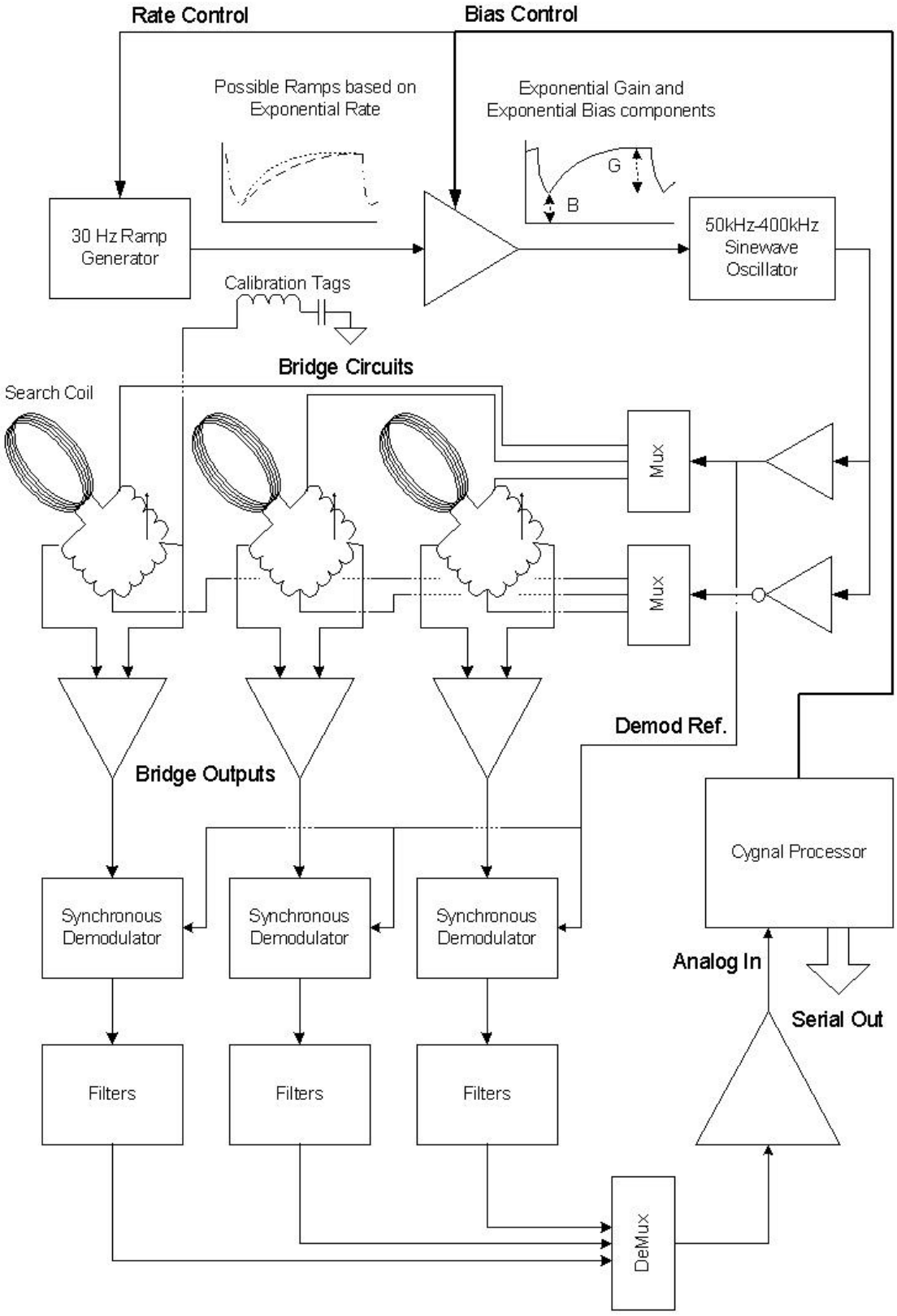


Figure 3-2: Block Diagram for Improved Swept Frequency Tag Reader

3.1 Processor Upgrade

The first modification was to the control section. The PIC 16C83/16C73 has been replaced with a Cygnal C8051F0005. The Cygnal processor offers more resolution in an improved 12-bit analog-to-digital conversion and much more capacity with 32k of code memory, 2.4k of data memory, and 24 assignable pins. It is also able to operate at up to 25Mhz, with fast code execution allowing a throughput of up to 25MIPS. These features allow for a number of upgrades to the board. The first improvement enabled entirely by the Cygnal, is an improvement, in sampling. Using the Cygnal, the tag reader is able to sample at up to 40kHz although it generally samples at 20kHz due to limited memory for baseline sampling (as described later). This sampling is also more sensitive and accurate, as the Cygnal provides 12-bit A/D conversion over the earlier 8-bit conversion of the PIC 16C73.

3.2 Dynamic Baseline Sampling

The next improvement, also enabled entirely by the Cygnal, is addition of a multi-point dynamically sampled baseline. The analog output signal provided by the inductive sensing is unfortunately nonlinear as a function of sweep frequency due to imbalances in the inductive bridge and noise susceptibility at different frequencies. Previously this had resulted in the suppression of much of the baseline, as a constant threshold for detecting a tag had to be based on the highest point anywhere in the baseline. The result was a loss in sensitivity as many tags had to move far above the baseline in order to pass that threshold.

With 2304 bytes of memory, the Cygnal is able to take 900 samples of the baseline. The threshold for determining whether a tag is present is based on the difference in the output signal

and the stored baseline at the tag's resonant frequency. This enables detection of a tag as soon as it brings noticeable distortion of the field beyond what might be expected by noise. The slight drawback of dynamic sampling is that since the Cygnal is only able to store roughly 900 samples, it places a limit on the number of samples that can be taken per sweep, which is far less than the limits determined by timing. This can be offset in three ways; the first is by simply sampling at a higher rate when checking for tags that are above the stored threshold while remembering the slower sampling rate of the baseline. A second would be by storing only 8-bits of baseline resolution. A third would be upgrading to a Cygnal chip with more writable memory, an option that has become available since the initial selection of the chip. The first technique of under-sampling the baseline is actually used right now. Checking for tags is done at a sample rate of 20kHz, while the baseline is sampled at half that speed. This actually does not impact the performance significantly, since in order to intelligently predict noise, the baseline threshold at a given point is set based not only on the given sample point but also on the neighboring samples.

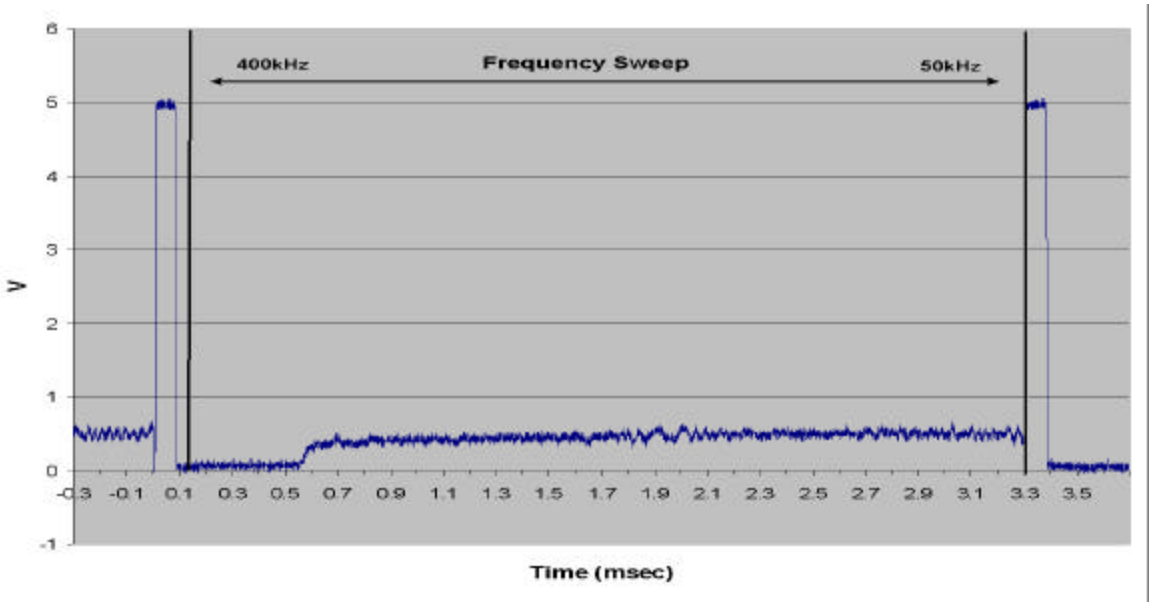


Figure 3-3: Swept Frequency Output Baseline

3.3 Frequency Drift

The next problem encountered with the original swept tag reader was frequency drift due to temperature. As discussed by Hsiao[1], it was found that the tag reader remains relatively stable over time, but can be significantly detuned by changes in temperature. Primarily due to the power transistors driving the search coil, the tag reader generates a fair amount of heat. It is also often run in an enclosed area, which resulted in the tag reader having to stay positioned for a number of hours before it was considered stable and final calibration could be done.

Frequency drift causes two problems in the system. The first is that the computer can lose track of which tag is which. The control processor does not relay the actual frequency at which a tag is detected, but rather the time relative to the beginning of the sweep. Thus, if the sweep shifts so that it is driving a resonant frequency at an earlier point in the sweep significantly enough, the host computer may mistake that tag for the adjacent tag with a higher resonant frequency. Although it might be possible to compensate for this by tracking the apparent movement of tags, there is an even worse second problem. If the drift is particularly severe, a tag may drift completely out of the sweep. As the magnetic sensing field is no longer driven at that tag's resonant frequency, that tag will no longer be at all detectable.

The solution implemented to fix frequency drift is twofold. In order to ensure that the second problem, the loss of a tag due to frequency drift, does not occur, the processor uses its crystal-driven programmable counter array, PCA, to keep track of the total number of wave cycles in a sweep and through this, provide dynamic compensation to maintain the original frequency sweep at which calibration was performed. Although minor drift will occur throughout the circuit, the prime drift derives from temperature instability in the oscillator and its control circuitry. This is problematic, as they are responsible for the generation of the curved exponential ramp that shapes

the sweep (see Figure 3-2) and the frequency-to-voltage characteristic. The exponential curve is generated using a 555 oscillator and buffer circuit, a major offender in producing drift, to charge a capacitor while high and then letting it decay naturally through a resistive load. The resulting curve is then buffered and amplified, providing control over the bias and gain. The final exponential curve is then used as the voltage control for an XR2206 monolithic function generator that produces the actual sinusoidal drive. This XR2206 is another major temperature sensitive component.

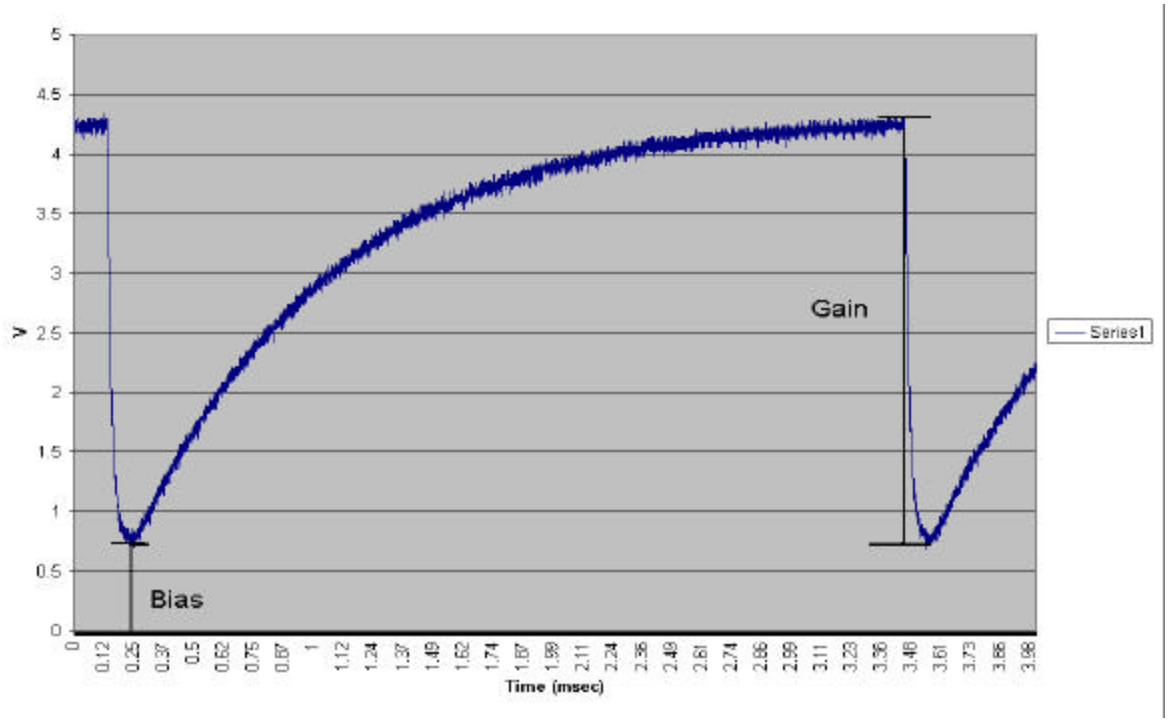


Figure 3-4: Exponential Curve for Sweep Generation

As the frequencies shift, so do the number of wave cycles. The Cygnal PCA counts the wave cycles in each sweep,and as the time interval of the sweep is fixed, this provides a metric for the span of the frequency sweep. In order to counter the drift, an AD5220 128-position

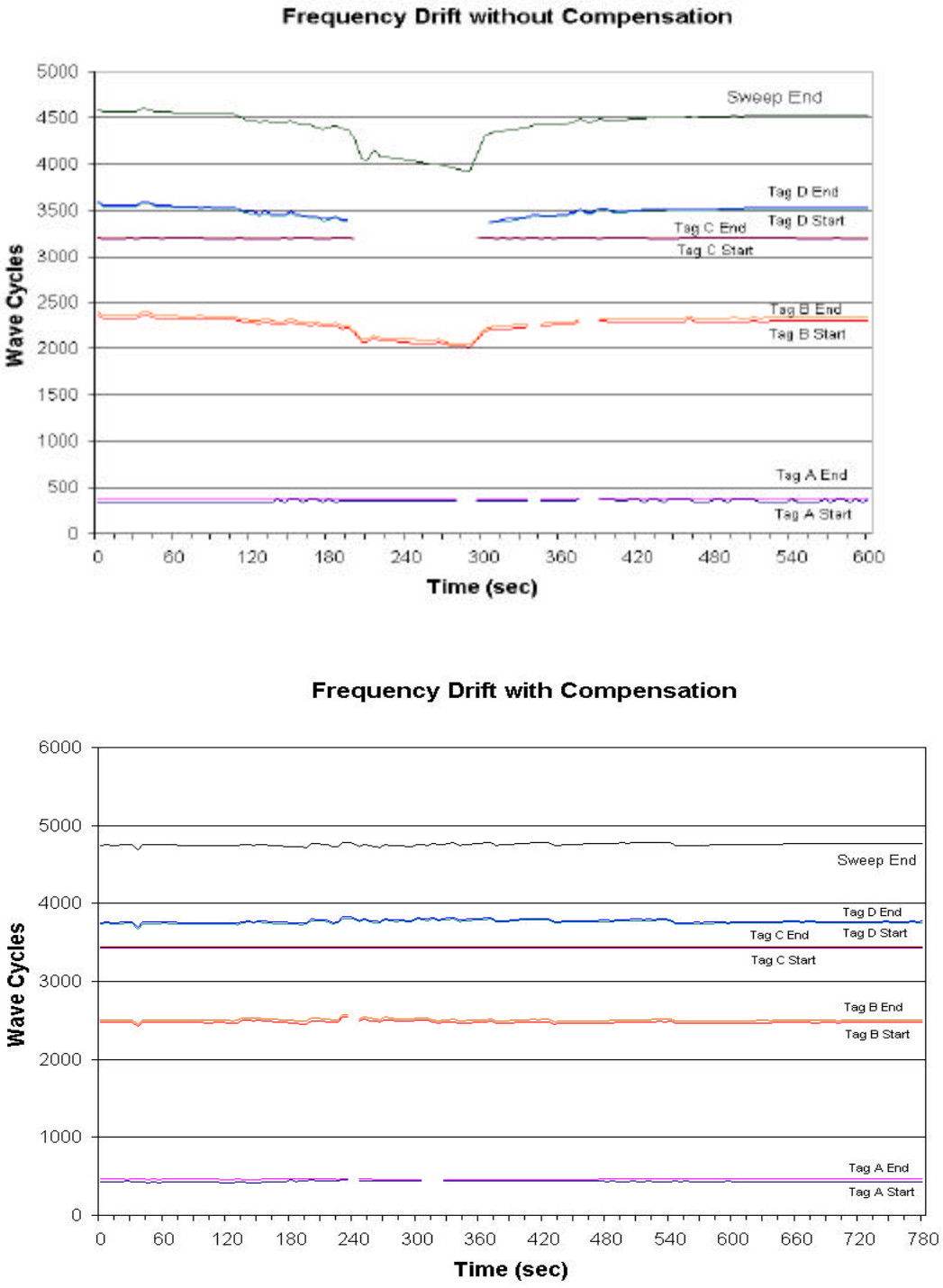


Figure 3-5: Comparison of Frequency Drift With and Without Compensation
Tests were performed used calibration tags A-D The temerature started at around 75 degress F, slowly rising 30 degrees to around 105F. Heat was removed at 300 seconds. Tags are switched so that loss of data from a tag is generally due to it being switched into the ciruit after the tag's resonant frequency has been driven.

incremental/decremental digital potentiometer has been placed in the circuit to dynamically manage the bias on the exponential curve driving the frequency generator. If the number of cycles differs more than 35 from the average number of cycles during initial calibration (there are typically 4000-5000 cycles per sweep), the Cygnal will trigger the digital potentiometer to compensate by raising or lowering the exponential bias voltage as appropriate. One step on the variable resistor will cause a jump of roughly 50 cycles either way. Figure 3-5 shows the frequency drift without compensation compared to the experienced frequency drift when using the variable resistor to provide dynamic stabilization. As the figure shows, although the number of wave cycles for a tag may vary, it remains within roughly +- 50 cycles throughout the compensated sweep. This is as much as $1/10^{th}$ the variation as occurs without compensation. The higher *Q* tags have a width of roughly 40 cycles.

It may be noted that the Cygnal processor offers two 12-bit DACs that would be capable of controlling bias and gain directly instead of using additional hardware in the form of the digital potentiometer. As the DAC's are 12-bit, they offer the benefit of finer tuning over the 128-position AD5220. However, the DAC's suffer from two drawbacks. The first is that as they are on the Cygnal processor, the DAC's are only able to provide between 0-3 volts. This could be overcome by stepping up the voltage or adding a bias to gain the full 5 volt range, but this obviously adds additional complication. More importantly, using the DAC requires more program overhead and thus more processing time. As the frequency drift is continuous, the incremental step response provided by the AD5220 is exactly what is needed, and controlling it only requires setting one pin, The DAC however would require reading, writing, and modification of two locations in memory. Sampling of the sweep can only begin after the Cygnal has completed sweep processing, regardless of whether the sweep has actually begun. Presently, this is a serious time constraint and there is

little to no processing time to spare. In future, if there are spare clock cycles or the need to jump across larger intervals, the DAC may prove a better choice.

While dynamic compensation for drift works quite well on its own, a second method for drift measurement is included. This is done using four on-board reference tags spread throughout the range of swept frequencies that can be coupled into the circuit near the inductive bridge. This allows any movement in the sweep location of these known tags to be used to by software to predict drift in other tags. The location of these tags is determined during initial calibration, and then during normal operation they are periodically switched into the circuit for cross-checks. Although largely redundant, the reference tags provide an independent monitor of the reader's performance. It should be noted that in Figure 3-5, reference tags are lost at several points, as indicated by the gaps in the data curves. This occurs for two reasons. The first is that only one reference tag is coupled into the inductive bridge at any one time. As such, if a tag drifts significantly, it may be coupled into the bridge after its resonant frequency has been passed. This is the likely cause for the temporary disappearance of the fourth reference tag in the uncompensated half of Figure 3-5. This difficulty can be corrected by altering the control to switch a reference tag in for an entire sweep rather having all four tags share only one. The second reason a tag may fail is due to the discrepancies between coupling strength. As these resonance tags connect directly into the inductive bridge, they can couple extremely strongly. Consequently, finding tags that do not couple overwhelmingly can be difficult. As a result, as shown in Figure3-6, the outside two tags couple far less than the inside tags and actually have a small enough magnitude that they could be lost due to noise combined with the high software threshold attached to calibration tags. This imbalance could be addressed by coupling the calibration tags via an appropriate attenuation network. The double-peaked tag signatures of Fig. 3-6 also indicate that

these tags are coupling in an inverted fashion as they are coupled into the opposite leg of the bridge.

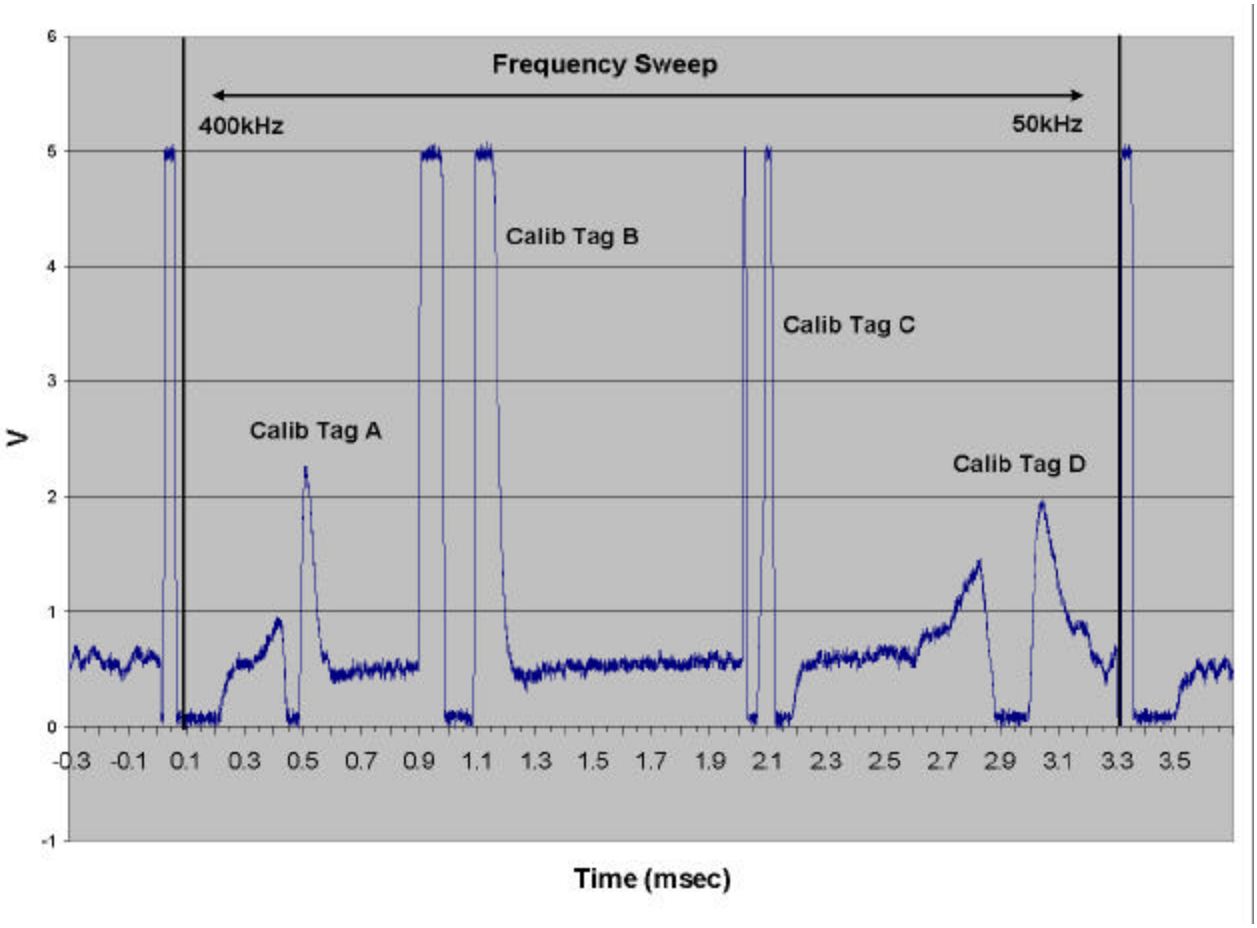


Figure 3-6: Sweep Output with Calibration Tags

One last note about frequency drift compensation methods and the sampling of the baseline is that in order to get initial values, the tag reader must start out in a calibration state. Upon initializing the board, the user can set the board in a halted mode, where the drivers will run, but the board will not be processing the data. This is to allow the user to manually tune the board without having the board try to automatically counteract any changes in the sweep. Once the board has been tuned, the user flips a switch and the board will perform initial calibrations. In order to get a good baseline and wave cycle count, the board will sample and average across 300 sweeps. After

these have occurred, the board must also place the calibration tags. The final result is that the board now takes roughly a minute after being lifted from the halt state before it is ready to start processing tag interactions.

3.4 Alterations to Inductive Sensing Circuitry

The third functional area of the board, the inductive sensing, has also been modified in order to take full advantage of the sensitivity and baselining now provided. Without a fixed baseline and with a less sensitive A/D, the earlier board required more analog gain. This is no longer necessary. As such, rather than using 6 amplifiers, the new board needs only 4 to achieve improved sensitivity and much wider dynamic range.

There is one other addition to the board, namely the addition of multi-axis capability to resolve larger areas and spatial coordinates. One of the truly interesting regions for exploration with the tag reader is developing multi-coil geometries. Depending on the geometry, this often involves cycling through multiple drives. A six-coil cube geometry was built previously[1] that used one frequency sweep that would be cycled across three pairs of coils to make a more uniform field successively spanning three orthogonal axes. This means only two coils were driven at any time. The implementation of this system was done by using 6 complete tag readers with frequency sweeps disconnected from the final amplification and bridge stages. The frequency sweep from the master board was then cycled by programmable switch to drive the disconnected amplification and sensing. The final detected output from each board was then routed through an analog multiplexer and returned to the master board for processing by the PIC microcomputer.

Figure 3-7: Old 6 Coil Tag Reader

As seen in Figure 3-7, this implementation is not only very large and ungainly, but also well over 1/3 of the circuitry is redundant or never even used. In order to enable the creation of a more compact multi-coil reader and also reduce repetitive circuitry, the new tag board is capable of running up to three coils non-concurrently. The tag reader uses a fast, high-current MAX352 IC precision analog switch to pass the completed frequency drive to one of three inductive bridges. This removes all redundancy within the frequency sweep generation and drive mechanism and means that all three coils are guaranteed to be driven by the same signal. As outlined in Figure 3-2, the three signals are then switched back together through an analog multiplexer just prior to the last

gain stage . This provides for the output phase and gain of each coil to be individually set. The one drawback is that the MAX352 analog switch is not entirely capable of handling the large current draw at the low end of the frequency sweep. Under roughly 90kHz, there thus begins to be a noticeable depreciation in drive signal. Lower frequency tags will still be detected but their dynamic range may be slightly more limited. If running with just one coil, this can be avoided by bypassing the MAX352. The end result is that without significant loss of capability, a six-coil reader can now be built with just two boards instead of six, bringing the size of the six-coil reader down from the size of a large desktop computer case (see Figure 3-7), to slightly more than a laptop.

Chapter 4

Early Tag Reader Applications

A number of applications have been developed using the swept frequency tag reader. They range from early simple interactions meant to demonstrate the bare functionality of the board to the more conceptually complex most recent application *Musical Navigatrics*. Although the tag reader has significant potential for usage outside the musical domain, the entire repertoire of applications to date have had some significant sonic output although almost more by coincidence than design.

The earliest application for the tag reader was implemented by placing the coil beneath a Lego board and embedding tags into Lego blocks[1]. Each tag had a corresponding tone that would be generated if the tag was placed into the reader's magnetic field. This implementation provided for a very basic demonstration of the tag reader's capability to sense and distinguish between tags. More complex interactions were also introduced by adding a multi-tag object and a variable frequency tag. The multi-tag object used three LC tags placed orthogonal to each other to determine the orientation of the object. This works because the magnitude of response to a tag is proportional to both distance and orientation. As the object rotates, different tags are rotated perpendicular to the magnetic field, changing the relative signal strength between the three tags. The variable frequency tag consisted of a magnetostrictor with a small magnet behind it. Moving the magnet would produce a continuous, detectable change in the response frequency.

4.1 Tangible Media and musicalBottles

Hiroshi Ishii and the Tangible Media Group at the MIT Media Lab have implemented a second application. Their work has involved the idea of taking the everyday inert physical object and introducing digital meaning into it while retaining the object's conceptual model. Tangible Media's application is a bottle that releases atypical contents when the bottle's stopper is removed and the bottle is opened. The project has explored a number of "contents" and interactions beginning with musicalBottles[6] and later bottlogues[7] and genieBottles[8]. Each implementation consists of three bottles placed on a table that doubles as a tag reader. With musicalBottles, opening the bottles will play one of the three lines (piano, violin, or cello) from Edouard Lalo's "Piano Trio in C minor, Op.7, while opening a bottle in bottlogues releases a single character's part in a three person narrative. GenieBottles controls the intertwining stories of three genies trapped in their bottles. Each interaction explores the physical nature of the bottles as a container and the digital controls created by them, letting the user realize the seeming impossibility of releasing sound from an opened bottle.

Figure 4-1: Tangible Media Group's musicalBottles

MusicalBottles and its successors all use the tag reader in the same hardware setup. In order to detect when a bottle is opened, a tag has been built into the mouth of the bottle. The cork

stopper has a ferrite core embedded into it while the bottleneck has coil wrapped around it. Thus, removing the cork (and with it the ferrite core), the bottle's tag is detuned. This is then detected by the tag-reader and passed on to a computer that interprets the data and uses a state machine to decide when to trigger the appropriate sound output.

4.2 *Musical Trinkets*

The third primary application, Musical Trickets [1][9][10], was done entirely by Kai-Yuh Hsiao and Responsive Environments Group. It is also based on using music to demonstrate the tag reader's capabilities. In *Musical Trinkets*, 20 tags have been placed in 16 objects, which then create and modify a musical soundscape based on their behavior in the sensing field created by an search coil connected to the tag reader and set into a table. The musical interaction is built around a set of melodic musical tones and a harmonic progression, with additional musical effects that modify and add to these tones and harmonies. The “trinkets” themselves are predominantly Halloween toys, with the result that the interactive objects ended up being mostly ghosts and goblins and a random assortment of other small toys, all embedded with tags. The main music generation tags are the ghost rings and the goblins. These trigger the musical tones and harmonies. The 5 ghost rings are worn on the fingers while the hand is held over a sense table with the search coil set in it. As the fingers are bent, the rings change orientation and become aligned to the sense field, thus triggering individual musical notes. It is somewhat similar to playing a piano. The goblins, meanwhile, control the harmonies. Individually, each of the 3 goblins triggers a different chord. As additional goblins are added or removed, *Musical Trinkets* uses a state table[1] to determine what chord is played next. The additional tags modify the musical texture provided by these 8 tags.

Figure 4-2: Tagged Objects used in *Musical Trinkets*

When added to the sense field, a Pikachu will add a twinkling of semi-random notes. The pitch region of this twinkle is determined by a Pikachu's proximity to the sense table. A pig modifies the musical lines by adding vibrato proportional to proximity, while a dinosaur with a magnetostrictor instead of an LC tag acts as a switch, triggering a change in the lines' instrumental voices. A foot provides continuous pitch bend and discrete transposition based on proximity to the sense coil.

There are also more complex interactions with specially tagged objects. As in the initial Lego experimentation[1], there is a cube and now also an eyeball, each with three perpendicular tags from which to track distance and orientation. The cube is used to add and bend a low droning voice, while the eyeball provides a particularly interesting interface as the tags coordinate to control various parameters on a Lexicon effects synthesizer. Lastly, a variable tag once again appears, this time as a Pez candy dispenser which fades in a choral sound based on proximity and an orchestral overlay based on the degree to which the tag is detuned. Detuning of the tag is

achieved by pulling up on the Pez dispenser head, which moves a ferrite core placed through a coil in order to chance the inductance of the tag. A cone can be used as a switch to change the Pez instrumental voicing.

Musical Trinkets has been shown at a number of conferences[9][11][12] and demos around the world to thousands of people. General response is very positive, surely due in part to relatively clear and easy to understand interactions, as well as an enjoyable and unique interface. However, *Musical Trinkets* is, as the name suggests, a fairly trivial musical interaction. Although it serves its purpose well as a demonstration tool for the tag board, the tagging interface, and the potential of tags as a musical controller for both solo and group improvisations, it lacks significant substance. It is limited in its usefulness for the real creation of complex music. Also, while some of the tags, for instance the effect-introducing eyeball, do make clear use of the continuous real-time capability of the tag interface, the exploration of the continuous nature is generally limited and fails to deeply explore and make strong use of the sound controlling potential that exists.

The data used to generate the sound controls in *Musical Trinkets* is also used to produce pleasant interactive graphics using OpenGL[1]. The graphics are projected from beneath a table onto a piece of translucent plastic that forms the table top inside of the coil. This provides for direct graphical user feedback. The graphics can also be rerouted for display on a bigger screen or other surface. They are generally coordinated to match either the physical aspect or the sound aspect of an object. For instance, rolling the eyeball around the table triggers the graphic of an eyeball rolling around the table, with the graphic rotating based on the actual rotation of the real eyeball object.

The initial graphic consists of a flat gray disk on a black background. Adding tags will either modify the disk or add additional graphics that move in relation to the disk. The two primary

Figure 4-3: Multi-user Interaction with the *Musical Trinkets*

musical tools, the ghost rings and the goblins generate bouncing colored balls and fade in individual colored spotlight-like triangles, respectively. The Pikachu adds swirling diamonds that spin around the disk, while the pig causes the bouncing balls from the ring to shimmer complementing the vibrato it adds aurally. The dinosaur flips the disk over, and the foot changes the perspective, essentially zooming in or out. The eyeball adds a graphical eyeball that rotates along with the real one. The cube presents probably the most interesting graphical response. As the cube is turned and flipped, the entire graphic rotates and flips along with it. The graphics associated with tags such as this provide a clear indication of the tag reader's abilities to determine orientation. Of the remaining two tags, the Pez tag adds a blue ring to the gray disk and a second violet disk when detuned. Lastly, in some variations of *Musical Trinkets*, the cone acts as a graphical switch between the tag's graphic described above and musicalCreatures, a music-

responsive graphic developed by Marc Downie from the synthetic characters group at the MIT Media Lab[13].

For more information on *Musical Trinkets* including video excerpts, please visit http://www.media.mit.edu/resenv/tags.html.

Chapter 5

Musical Navigatrics

Musical Navigatrics is the most recent and most complex application developed for the tag reader. It was designed by the author an attempt to create a comprehensive and exciting musical instrument using the tag reader, and in the process explore the various potentials of the tag reader as an expressive musical interface. It is the next step beyond *Musical Trinkets*, adding depth and musical creativity. There are three conceptual areas that *Musical Navigatrics* explores: using the tag reader to implement a free space musical instrument, its functionality as an effects controller, and adaptability of the discrete nature of moving tags in and out of the magnetic field to develop a basic sequencer, enabling complex tracks to be recorded, overdubbed, modified and accessed during performance. These areas provide the foundation for the three main components required to build an expressive song: notes, timbre, and order. The ability to combine these three musical aspects together provides for a large amount of musical potential and depth, as well as real practicality towards developing and composing a performance. All mappings of tags to music are thus geared to give full musical flexibility to the user within the limits of musical hardware and software as well as the tag reader itself. Considering the fact that most of the time, for meaningful musical production, *Musical Navigatrics* would be played by one, possibly two people, each with only two hands available for manipulating objects, mappings were chosen to enable as much control as possible considering that generally only one or two tags may be giving dynamic input at any given time. It is this aspect that greatly enhances the need of the sequencer. The sequencer

allows the development of multiple voices as well as the addition of effects on top of already played lines. The choices of sounds and effects were also selected to provide for the most interesting but also discernable range of output. The end product is a substantial expansion of *Musical Trinkets* into a musical instrument and expressive free-space means of musical production.

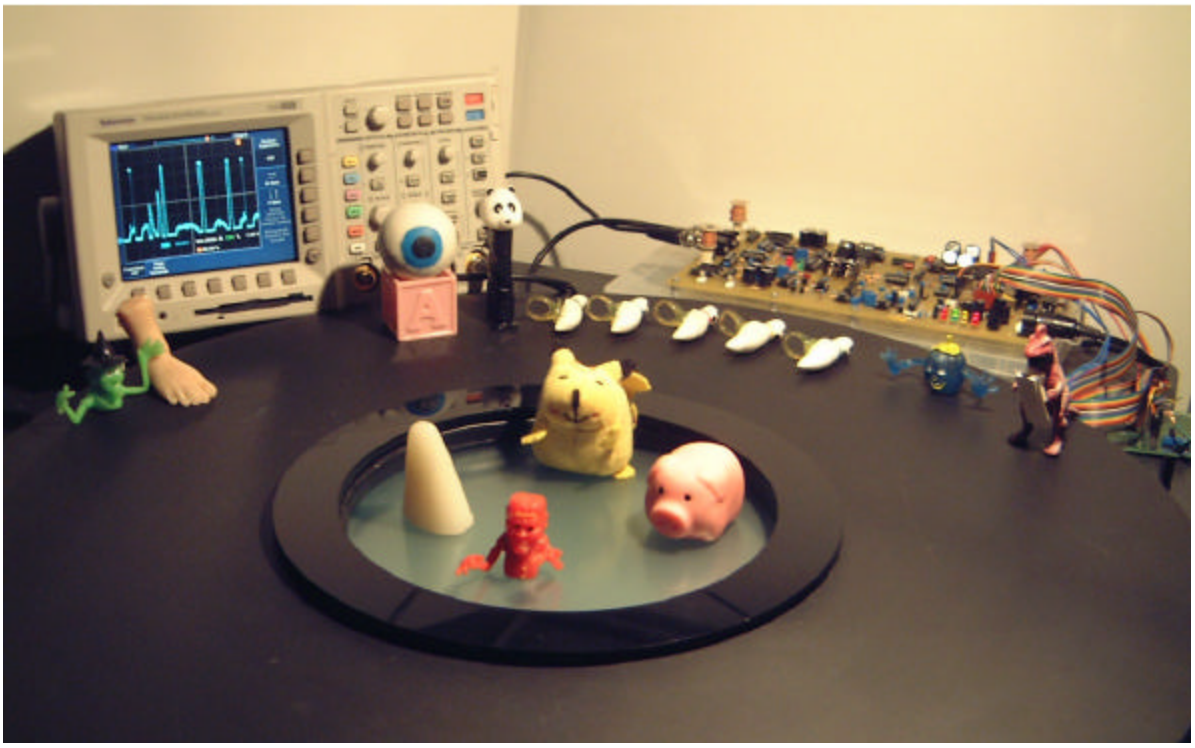


Figure 5-1: *Musical Navigatrics* Setup

5.1 Technical Implementation

In starting the new application, it was decided to move away from implementing MIDI mappings using C++ and Rogus[14] and instead use a more powerful, better supported program that is more directed towards musical applications. *MAX*[15] describes itself as "a graphical music programming environment for people who have hit the limits of the usual sequencer and voicing programs for MIDI equipment."

This describes just what was needed for the task. *MAX* provides for simplified MIDI mapping. However *MAX* is currently written only for the Apple Macintosh platform and while Macintosh's are generally very good for music applications, they lack non-USB serial support. The current tag reader only outputs data on a DB9 serial cable. It is, however, easy to route MIDI to a Macintosh using one of the many MIDI-USB adapters. Thus the MIDI stream for *MAX* is produced by taking the serial data produced by the tag board used with the tagged objects from *Musical Trinkets* and processing the data using a modified version of Kai-Yuh Hsiao's earlier PC based *Musical Trinkets* application. As the original *Musical Trinkets* was already generating MIDI signals using the tag reader, it was fairly simple to adapt it to pack the information regarding present tags and their signal strengths into MIDI data. The MIDI stream is then sent to a Unitor MIDI interfaces, as also used in *Musical Trinkets*, which distributes it to the MIDI input for *MAX*, in this case, a MIDIMAN MIDISPORT 4 connected to an Apple G4 Cube.

In *Musical Navigatrics* the modified *Musical Trinkets* program sends MIDI notes via a MIDIMAN. The MIDI values are reassigned, however, so that each MIDI note value corresponds to the presence of a different tag, while the note velocity relates to the signal strength. This standard tag information is sent on MIDI channel 0. Channel 1 is used to relay any additional tagging information, which at present is restricted to the variable-frequency Pez tag. For this tag, its overall presence and strength is still sent via channel 0, but a corresponding note value on channel 1 is used to relay any shift in the tags frequency.

Once the MIDI data has been received by *MAX*, it is translated and remapped to run the music production side of *Musical Navigatrics*. At present, there are two versions of *Musical Navigatrics*. These two versions are functionally the same, only they use different sound production techniques. In the initial version, MIDI out messages from *MAX* were sent to hardware

synthesizers. Musical notes were produced using an EMU Proteus 2000 Sound Module while effects were handled either by the EMU or a Lexicon MPX-100 Dual Channel Effects Processor. All sequencing was implemented from scratch in *MAX* using the *MAX seq* function for actual recording.

Version II removes the need for sound generation hardware by using Propellerheads *Reason* software[16] for sound production. MIDI messages, both note and control, from *MAX* are internally passed to *Reason*, which redirects them among its various virtual rack devices. Additionally, all prerecorded sequences, such as drum loops and neutral figuration, are created and handled within *Reason*. Real-time sequencing and sequence coordination is still being performed in *MAX*.

5.2 Musical Implementation

Like the three conceptual areas of *Musical Navigatrics*, the tags themselves were divided up into 3 corresponding categories: note production tags, expressive effects tags, and control tags. Note tags are any tags that trigger a sound. Effects tags are used to modify the note voices and the control tags are used to store information about the movement of the note and effect tags.

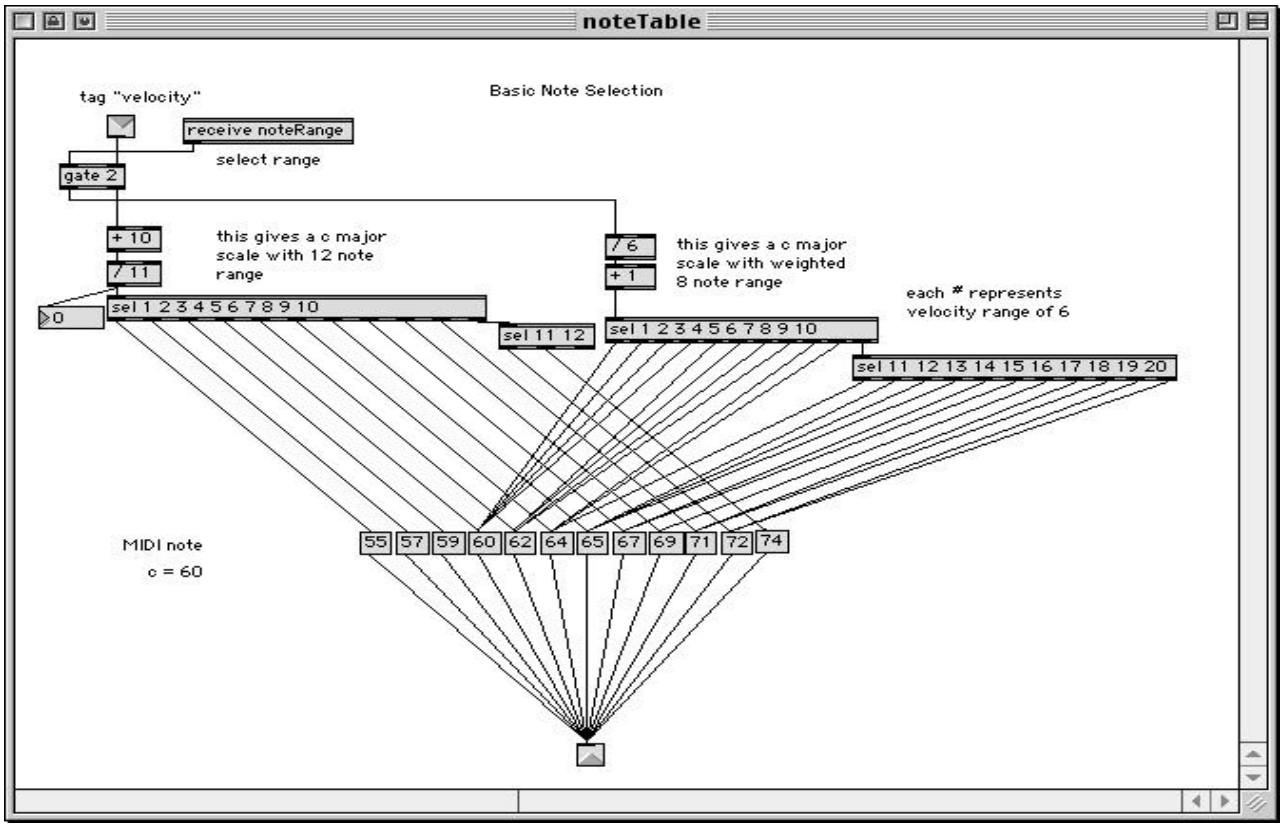
Note Tags: There are 5 note tags. The first three (the red goblin, the green goblin, and the blue goblin play) respectively, a soprano voice, a tenor voice, and a bass voice. The note played corresponds to the signal strength of the tag, so that the tag's distance from the table more-or-less decides the played note. Moving the tag up and down vertically over the table will step through a discrete C Major scale. The farther the tag is from the table, the higher the note produced. All the tags have either an 8 or an 11 note range (selected in *MAX* by the user) going from either C to C an

octave up, or the V below C (G) to the D an octave up. These ranges, though somewhat small, were chosen to take the best advantage of the available dynamic range; allowing the user to pick from a

Tag Behavior In Field		Direct Sound Produced	Record (No "Play" present or voice not recorded)	Record ("Play" and voice already recorded)	Play (Voice already recorded)	Line Effects? (Other Voices Active)	Line Effects? (No voices active)	Master Effects
Voice Tag: Any Goblin	Not Present	None	No Action	Prev Action	Prev Action	No	Yes	Yes
	Active	Current Action	Current Action	Current Action	Current Action	Yes	Yes	Yes
	On table	None	No Action	Prev Action	Prev Action	Yes	Yes	Yes
Voice Tag: Dino. Or Pez	Not Present	None	No Action	Prev Action	Prev Action	No	Yes	Yes
	Active	Current Action	Current Action	Current Action	Current Action	Yes	Yes	Yes
	On Table	Max Value	Max Value	Prev Action	Prev Action	Yes	Yes	Yes

Figure 5-2: Voice Tag Interactions
"Action "is defined as any movement within the sense field and an "active" tag is any tag within the field not sitting on the sense table. The first record category refers to when there is no Play Pikachu present or, if it is, the voice has not already been recorded. The second record category refers to both the Play Pikachu being present and the voice having been recorded. The effects categories refer to whether an effect will be heard on that voice or not.

useful range of notes while not packing so many notes into the sense region that it becomes difficult to locate a particular note. Thus, the blue goblin playing the bass voice also offers a 5 note range, C to the V above (G), as the tag is not as strong as the others and has a reduced dynamic range. Bass voices traditionally jump around the scale less, so this is also somewhat appropriate. Lastly, due to the fact that the field strength is non-linear with positioin, the mappings are written to help linearize the playing region, as this makes a more intuitive interaction. This weighting, as well as the notes out, can be seen in Figure 5-3 the *MAX* note mappings. One last aspect of the voice tags is that if they are placed on the table, they will actually stop playing. As shown



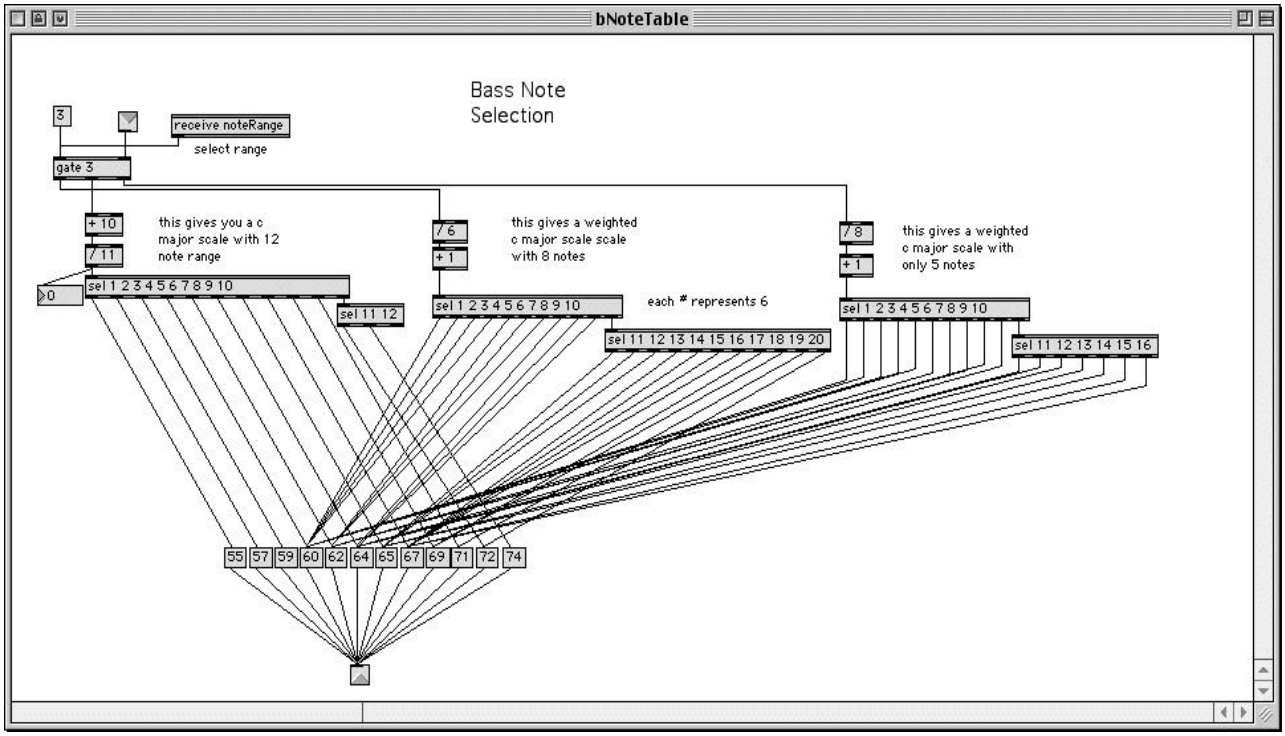


Figure 5-3: *MAX* Note Tables for Note Determination

In Figure 5-2, there is no note mapping for a goblin sitting on the table as this state is used to select a voice for line effects. This will be further explained in the section on line effects.

Figure 5-4: Note Tags

The other two note tags are less strictly notes, as there is not quite the same distance-to-note correlation. However, they functionally act as note tags, in that they trigger new sounds, can be recorded by the sequencer, and can be modified by the effects tags. The first is the dinosaur. Moving the dinosaur into the tag-reading field triggers a predominantly pre-recorded drum pattern. In order to provide the user with more than one drumbeat, the dinosaur takes advantage of the dynamic range of the signal strength provided by its proximity to the coil to select different rhythms. The closer the dinosaur is to the coil, the generally more aggressive and dense the rhythm played. There are seven different pre-recorded rhythms to choose from, with a variable number of random notes added, depending on the dinosaur's coupling with the coil.

The last note tag is the Pez dispenser. It controls either a chord or neutral figuration. When initially placed into the tagging field, it will play triad C-Major scale chords. As in *Musical Trinkets*, the Pez dispenser is a variable tag, so that raising the Pez head will turn off the chords and instead step through a selection of 4 pre-recorded neutral figurations with each additional head raising. Moving the Pez dispenser vertically over the tagging field table changes the chord or transposes the figuration. This time there are 8 possible chords/transpositions ranging from C Major to C major, but the order is reversed from the goblin voice tags, so that sitting the tag on the table will trigger the highest chord instead of the lowest. This allows triggering of the low C chord upon removal, a more typical progression than the other way around. Along with this, the chords are weighted so that the more common chords, such as I, IV, V, VI and I octave, have a large playing region and are easier to hit than less useful chords like III.

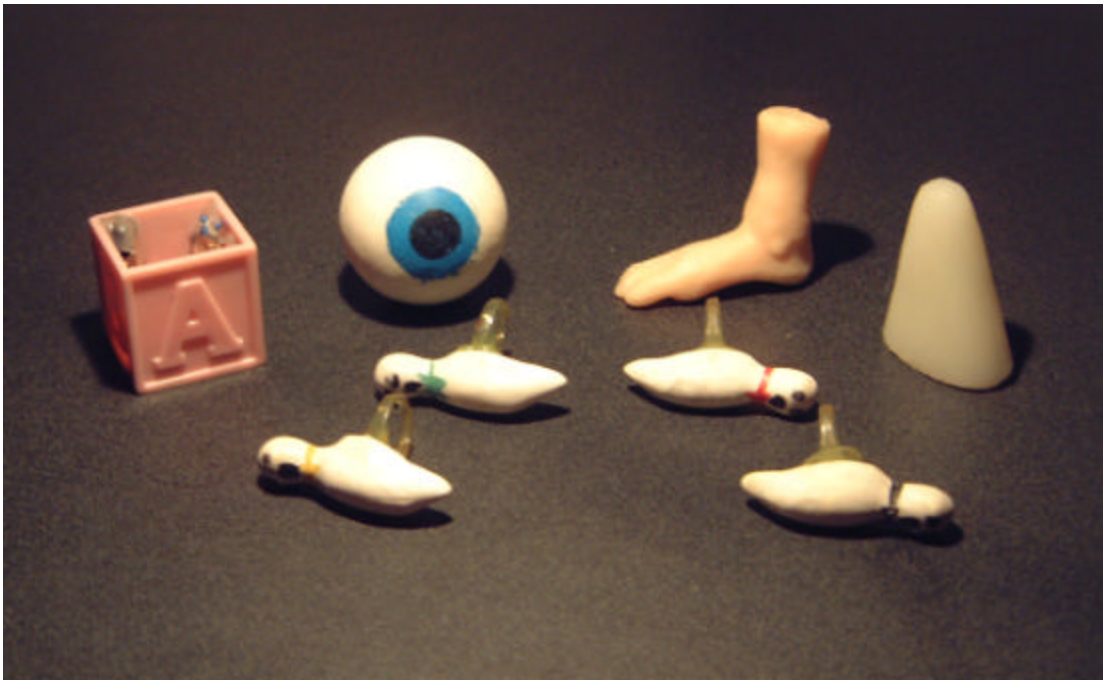


Figure 5-5: Effects Tags

Effects Tags: There are a total of 8 different effects objects controlling behaviors ranging from volume to transposition to filtering. The effects tags are divided up into two categories, line effects

and master effects, based on their scope. Regardless of their function, line effects are able to act upon and modify any given note tag or selection of note tags. Master effects meanwhile are more limited in scope to act on every tag or a specific non-changeable set of tags. While effects tags can be played and even recorded independently, they require a note tag or already recorded note-tag sequence to be heard.

Line effect tags control effects where the user can choose one or more note lines and control the sound of just that line alone. A line is easily selected by placing the corresponding tag into the tag field. As discussed in the section on note tags, placing a voice tag on the sense table will select it for line effects and will not actually play new notes. This is particularly useful when playing back recorded lines, as it allows a recorded voice to be selected for effects without playing over the recorded line. For instance, when *Musical Navigatrics* is playing back a sequence composed of a bass voice line and a high voice line, if the red goblin high voice is placed on the table, any line effect introduced to the sense field will operate only on the high voice. However as any tag dynamically present in the sense field receives effects, if the same line is played back with the red goblin still on the table but this time the green goblin middle voice is being actively played, the effect tag will now be applied to both the red and green goblins' corresponding voices.

Both the Pez figuration tag and the dinosaur drums can be selected in the same way for effects, but unlike the voice tags, they have no passive select state and will always play based on their proximity. If all note tags or no note tags are present in the sense field, line effects will operate on all note lines.

The available line effects are: filter cutoff frequency, filter resonance frequency, transpose and volume. Cutoff frequency and resonance are mapped onto the black ring and the red ring respectively. Their basic operation is quite simple, in that the magnitude of an effect is directly

controlled by the signal strength of the coupling between the tag and the coil. As the tags themselves are rings, they are not expected to be used in quite the same perpendicular up-down manner as many of the other tags. A more common mode of interaction is to wear the rings. The result is that the effect magnitude changes not only with distance from the sense table, but also changes as the hand moves and rotates through the field. Wearing multiple effects rings (there are also two master effects rings) triggers musical response to the shape of the hand and its full movement through space.

The Transpose Foot will transpose all selected voices (with the understandable exception of the unpitched dinosaur drums) up in relation to the C Major scale. There is no transposition when the foot is sitting on the sense table, but as soon as it is raised, it will move selected voices as much as a full octave. This provides the user the opportunity to introduce easy additional chord structuring. The Volume Cone unsurprisingly acts to control the volume of a voice. As an extension, the Volume Cone also acts as the mute cone. When the cone is placed on the sense table, it will mute all selected voices and slowly return them as it is removed from the from the sense field.

For the most part, master effects are effects that, due to the hardware or software implementation at the point of sound generation, are only available to act on all sounds. This is true for 3 of the 4 effects: the cube, delay ring and the distortion ring. The eyeball, on the other hand, rotates a set of effects through the three primary voices. Regardless of the actual effect, however, they are unable to act independently on one voice and thus are considered master effects. As a shown in Figure 5-2, unlike line effects, it makes no difference to the sound output what other tags are present. As long as there is sound being produced, a master effect will act on it.

Apart from their region of effect, the delay and distortion rings act much like the resonance and cutoff rings. The signal strength from the tags in the rings is simply passed on as a MIDI control signal and sent to the relevant device to add delay or distortion. The cube and the eyeball have much more fun and unpredictable effects. Again, as in *Musical Trinkets*, both of these two objects is embedded with three perpendicular tags. The cube uses its three axes to control three separate parameters of the attached Lexicon effects processor, adjust, mix, and effects level. In Version II, the cube alters panning as well as phasing and compression. The eyeball, rather than using the three tags to produce effects in coordination with each other, links each of its three tags to one of the three primary goblin voices and sends a set of effects to each voice based on how much its tag couples. As the eyeball rotates and rolls around the table, effects essentially rotate through the voices, altering one or two to voices at a time. The eyeball triggers a variety of effects: slight pitch modulation, decay modification and alteration of one of the filter envelopes. In *Reason* it once again causes slight pitch modulation, but this time also adds distortion and alters filter modulation. Regardless of the software version, the eyeball ends up creating a somewhat unpredictable but highly intriguing interaction.

Control Tags: Control tags run the sequencing capabilities of *Musical Navigatrics*. There are only three control tags, Play Pikachu, Record Pig, and the Tempo Ring, but they dramatically impact the usefulness and depth of *Musical Navigatrics* by letting the user move beyond controlling one or two aspects of musical creation and instead offering a whole realm of sound possibilities. The Record Pig and the Play Pikachu are the only two tags for which there is a simple on/off state. They work in conjunction to play and record the user's various gestures. In order to maximize the

creative capabilities and place as few restrictions as possible on the user, the movement of almost any tag can be recorded into a sequence.

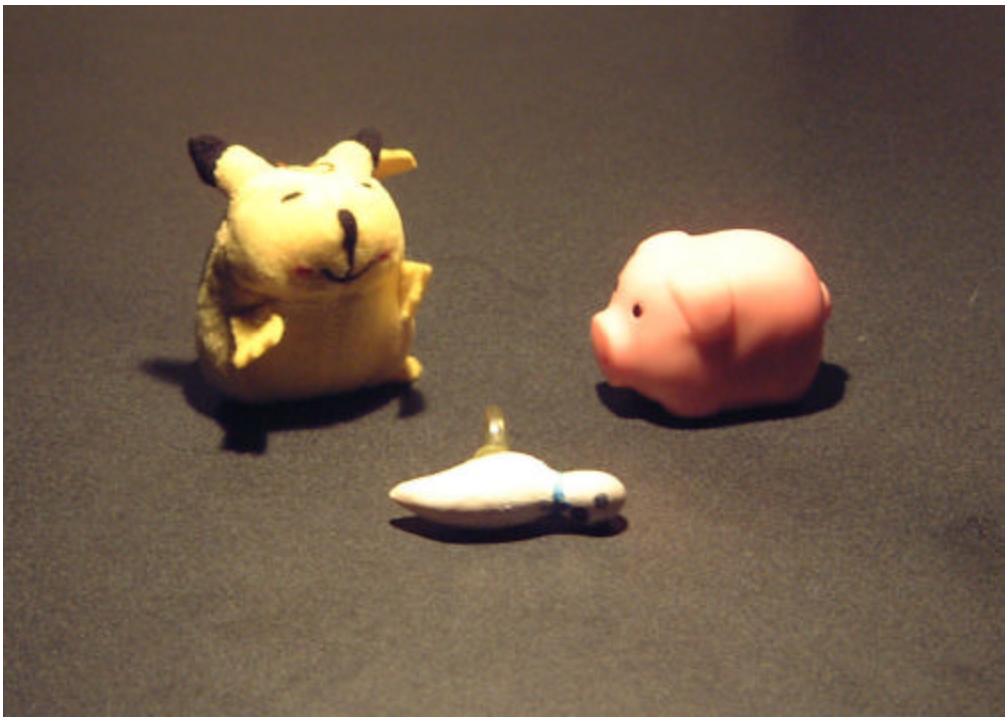


Figure 5-6: Control Tags

Sitting the Record Pig in the sense field will start the record process by triggering a drum pick-up intended to provide the user with the tempo and a bar line. The pick-up will start with the first beat after the pig is placed in the field, and recording will begin on the second downbeat (emphasized by a base drum). Once recording has begun, the movement of any tag in the field will be recorded for later playback. This includes changes in drum rhythm, playing of a voice or chord, or an even a line or master effect. The only exception is that other control tags are not recorded. However this does not mean they will not affect the recording.

The record function is loosely quantized. The level of quantization can be changed but must be done on the computer. The level of quantization ranges from a quarter note to a 32nd note.

Also, recording will continue until the end of the bar in which the Record Pig is removed from the tag field. A metronome can be turned on directly using *MAX* to aid a user to record in time.

Placing Play Pikachu on the sense table will begin playback of whatever was most recently recorded. *Musical Navigatrics* will only play one line per tag, however, so as indicated in Figure 5-2 if an already recorded tag is being actively played, its new motions will be heard rather than the recorded line. This is true for both recorded musical tags and recorded effects tags. As discussed in the section on line effects, sitting a goblin voice tag on the sense table will not supercede the corresponding recorded line, it just selects that voice for effects.

Adding the Record Pig to Play Pikachu allows for overdubbing. The already recorded line will be played back and re-recorded while any new tag movement will be added. Similar to normal playback, a specific already-recorded voice or effect can be overwritten by actively playing that voice or effect during recording. At present, there is no way to completely turn off a line other than mute it.

The last ring, the blue Tempo Ring, controls the playback tempo of all recorded lines, as well as the preset drum lines and neutral figurations. Moving the ring far from the table will decrease the tempo, while moving it close to the table will speed the tempo up. The active tempo range is 64-187 bpm. Tempo should not be changed while recording, as it will result in desynchronization of the timing between the record and play functions.

The last control functionality is the SET parameter. With tags such as tempo, resonance and the Drum Dinosaur, it is often desirable to save the value of the tag so that the value of the corresponding control can be chosen without having to leave the tag in the field. For instance, without the SET ability, any time the Tempo Ring is taken out of the tag field, it would be sensed while being removed, decreasing the tempo until the tag is out of the field, at which point the final

tempo is always very slow. However seeing as it is non-optimal to have to hold the Tempo Ring in the right place all the time, SET will do it instead. SET is done by taking the Record Pig and quickly passing it through the sensing field. The value of any tag in the field will be stored at that instant and played back as such. The tag being SET must also be quickly removed from the sense field or it will overwrite the saved value.

5.3 Evaluation of *Musical Navigatrics*

Musical Navigatrics has been shown at the MIT Media Lab during several and has been viewed by a large number of people. Although it has elicited a fair amount of interest and positive response adding depth to the system beyond *Musical Trinkets* has also made the interaction with the system far more complicated. Here the limited interactions offered through demos is unfortunately only able to allow initial impressions. *Musical Navigatrics* requires some practice to master, an effect that is coherent with the goals of this project, and as such, the application is presently not organized towards simple types of play such as found in quick demos. Evaluation of its functionalities are thus based not only on observed user interaction, but also on substantial personal experience.

MIDI Effects Controller: Of the three areas that *Musical Navigatrics* explores, it succeeds best as a MIDI effects controller. The continuous nature of the interaction allows gesture to become effect and through effect, expression. Effects rarely need to be precise, as they relate more to exploring a certain musical feel. *Musical Navigatrics* lets the user make the natural translation of feel to gesture in a comprehensible manner not easily found elsewhere. The most standard interfaces for controlling effects are small knobs, buttons, pedals, and keyboard mod wheels,

pressure and aftertouch. All of these are restricted to a small, if not tiny range of motion. Anyone who has gone to a concert of today's electronic music and watched the musicians bend over boards as they turn knobs can see how un-expressive and how un-related these interfaces are to the changes in sound they effect. They also make for a potentially boring live performance as the audience has little visual context with which to follow the musical gesture.

The lack of non-restrictive effects and expressive controller interfaces is a recognized problem. In the past 5 years, Korg and Alesis have both released interfaces that try to address this issue. The Korg *Kaos* pad [17][18] has been highly successful. It consists of a small rectangular touch pad that uses X-Y location to control effects. Although it is being used by a large number of major artists especially within the dance music community, and has been widely praised for its interesting effects and MIDI output capability, its deepest contribution is that it is an intuitive and easy to use music interface that lets the user move well beyond what is possible with simple knobs.

The Alesis *AirFx* [19][20] is a free gesture interface closer in usage to *Musical Navigatrics*. *AirFx* creates an infrared cone that is about 6" long. When the user places his hand in the illuminated field, AirFx uses the intensity of reflected light to infer the distance and the orientation of the hand, and use that to control a bank of 50 preprogrammed effects. The AirFx suffers somewhat from lack of MIDI output capacity and mainly weak effects, but it has still been well received due to its usefulness as a live interface as well as the ease and interest of interaction.

Another popular group of mainly non-commercial effect-controlling interfaces are those based on capacitive sensing[21]. There have been a number of different platforms developed around the idea of capacitive sensing to develop a free gesture interface. Probably the most famous and most successful is the Theremin[22], but in general, capacitive systems are difficult to control. They suffer from problems with stability and the need for calibration, along with the common free

gesture problem; they offer no tactile or visual interface. This lack of feedback makes capacitive systems very hard to learn. An experienced user can play them in a virtuosic manner but to the casual user, they are often too difficult and abstract to be useful. One way of making capacitive sensing more palatable is through the addition of visual stimuli, such as, Leila Hasan's *Termenova*[23], done in the Responsive Environments Group.

The recent interest these alternative music platforms have generated, especially considering the success of the two commercial devices, indicates just how effective *Musical Navigatrics* could be. The exciting thing about *Musical Navigatrics* though is that it offers more than these interfaces do. The *Kaos* pad remains restricted to a small 2D board and provides only limited improvement in terms of visual interaction. *Musical Navigatrics* is able to move beyond both problems. The *airFX* is capable of 3D interaction, but is restricted to one source of data and can be obstructed by smoke (common at least in the DJ community it is aimed at) or bright light. Both the *airFX* and capacitive sensing suffer from a not always clear responsive interaction and also lack any tangible reference. Even though it is the placement of the tags in free space that controls an effect, having something to hold is reassuring to the user and naturally enables easy immediate selection of different effects, an inate advantage of tag reader that can not be matched by any of the other systems discussed; they are unable to distinguish one hand from the next. Added to this is *Musical Navigatrics* ability to recognize orientation, such as with the cube and the eyeball. These two objects add the most interesting physicality, as they give the user the ability to really explore a 3-D soundscape, grabbing and twisting the object in physical space and aural space. The interface discussed so far does not even include the possibilities for the tag reader as a multi-coil effects device, which would enable multi-axis spatial diversity and control over additional degrees of freedom.

Musical Navigatrics performs very strongly as an effects controller. It is flexible, interesting, intuitive and provides a very natural reflection of the aural results of effects. The ability to pick from a variety of objects, move then in simple sensible manner, and hear understandable results is ideal. The visible nature of the movement makes it good for live performance. The only two drawbacks are the limited sensing range and the need for a hold function, two drawbacks that can be easily addressed.

***Musical Navigatrics* as a Tonal Musical Instrument:** Although *Musical Navigatrics* performs beautifully as a MIDI effects controller, it does not succeed as well as an independently playable instrument. Unfortunately, it suffers from the same problems as most free space instruments, in that it lacks direct haptic feedback. Without the user being able to feel the distinction between notes or even have a visual reference, it becomes difficult to teach the body just where in space specific notes are. Without this knowledge, it is difficult to control the *Musical Navigatrics* well enough to outline a song. No implementation of *Musical Navigatrics* will be able to completely overcome this problematic lack of direct physical feedback, however our implementation does offer some improvement over the standard free gesture situation. Firstly, the fact that changes in pitch are discrete provides enough aural feedback so that with a small amount of training, it is possible to develop some sense of where notes rest in space. The discrete nature also means that a pitch can be given a substantial area in space, making it easier to hit. Also, the table the tags are played over provides some indirect physical point of reference. By resting the elbows on the table, it becomes much easier to stably go to a desired pitch. If the "size" in space of a note is sufficiently large and the table is used as a physical starting point, it can become quite easy to locate notes and

play a song. Using the table and an 8-note range, the author was able to play basic songs such as the hymn "Dies Irae" in only a few tries.

Despite this success, *Musical Navigatrics* is still not particularly satisfactory as a pitched instrument. Part of this was the fact that in the present hardware, there is not enough sensitive spatial range in which to place enough pitches to create a particularly meaningful instrument. There are, of course, many instruments that have existed for hundreds of years that posses as few as 5 or even one or two notes, but these instruments are generally specialized, traditional, or percussive; all concepts that are not really goals in this project. At present, the space above the tag reader can be divided up into either 8 or 12 notes. With a bit of practice, the 8 notes is generally feasible to learn, but even just the 4 additional notes pushes the limits of what is easily playable.

Yet another stumbling block for *Musical Navigatrics* as a pitched instrument is the non-uniformity of the magnetic field in the plane over the table, which combines with the orientational nature of the coupling to make transverse placement and object tilt additional parameters that affect the sensed signal strength. While this property is well-suited to *Musical Navigatrics* functionality as a MIDI effects controller, it hurts its functionality as a discretely pitched instrument. As said before, free-space instruments are hard to play precisely as they requiring careful and difficult positioning of typically the hand in empty space. The fact that not holding the tags straight or not moving them in a strictly up and down motion, not to mention the fact that the magnetic field itself bends around the coil, all impact the signal strength and thus pitch selection. Hence, there ends up being many more complicated variables to learn for effective performance than just vertical height.

There is one last problem area for *Musical Navigatrics* as a pitched instrument: the lack of a rest. Unlike the other problems discussed, this problem is not inherent to the tags and the tag reader but rather the application. Not having a rest, breath, or control over a note's decay was a

significant oversight in planning. It severely restricts the expressivity of the instrument as well as making it difficult to jump notes. It is, in effect, impossible to repeat a note and very hard to form a line without giving it an undesired tail due to either triggering other pitches while removing the tag from the sensing field or moving another tag while trying to end the line through other the movement of other tags (such as muting the line or stopping recording). This problem could be somewhat solved by adding a "rest" tag, which stops a note when it enters the sensing field, but part of the reason this has not been implemented is that it requires the use of both hands in strict interrelation and precise conjunction that is not entirely intuitive. (The "volume" tag could theoretically be used as a "rest" tag, but in practice, it requires much too large a motion to be effective). A slightly more promising solution is to add a "decay" tag. This would be fairly easy to implement, as it just requires mapping the synthesizer decay envelope to a tag, but suffers in that it results in an even more complicated interaction between controlling pitch and note length. On the positive side though, being able to control the decay adds a significant degree of expressivity.

Both options do require a significant degree of hand coordination. This is a definite obstacle, but at the same time, such a “damping” technique is used with great success in other instruments. For instance, in Balinese gamelan, one hand is used to strike a key on a metallophone, while the other hand lags behind by a note and provides characteristic damping. This is an even more difficult interface, yet can still be learned fairly easily. More difficult to overcome is the fact that requiring a "rest" or decay tag results in no free hand to add other expressive or control gestures.

One last option that requires modification of the tags themselves is to add a button or something similar to the tag that either opens or shorts the LC circuit, destroying the resonance.

This is still a somewhat complex coordinated interaction but as it is confined to one hand, should be not only easier than a two-tag method, but also leave the second hand free for other interactions.

In the end, *Musical Navigatrics* succeeds only somewhat as a pitched instrument. The difficulty of consistently hitting a note, combined with the lack of a rest, make it of limited appeal. This does not mean that it is of limited use; the Theremin is a successful instrument although only a few experts can really play it meaningfully. Adding a decay tag may alleviate part of the rest problem, but has its drawbacks, including reducing the ease of use. Probably the best solution is to rethink the way *Musical Navigatrics* can be used as a pitched instrument. Rather than trying to turn it into something like a vertical keyless keyboard, why not use it as something more like a curve generator or even more promising, use it to control an arpeggiator. Although the possibilities in this area are potentially vast, one simple template would be to employ the "Matrix" template in *Reason*[24]. Although the Matrix can be used for far more complex sequencing, an initial interaction is to draw interesting tonal curves. The Matrix, by default, makes all notes the same length and inserts breaths between them. A similar mapping would be very well suited to *Musical Navigatrics*. It may lack the apparent depth and capabilities of the original goal of this project, but it remains an interesting, fun, and even useful interaction. Interfacing with an interesting arpeggiator has the potential for far more complex interactions.

***Musical Navigatrics* as a Sequencer:** In the entirety of *Musical Navigatrics*, there are only two discrete on/off tags, and they are both control tags. The amount that these tags are used in practice belies just how important and useful these tags are. They are easy to use with fairly clear effects. But at the same time, they remain very limited. Compared to a commercial sequencer, *Musical Navigatrics* offers an odd balance of unusual capabilities versus a distinct lack of

capabilities. It is the unusual features, however, that demonstrate the tag reader's strengths and uniqueness, while its limitations demonstrate the limits of the implementation and potential room for expansion.

When thought of as a solely a sequencer, *Musical Navigatrics* performs weakly. Part of this is due to the fact that fully programming a powerful and thorough sequencer is a sizeable task, far beyond the scope and needs of this present project. But *Musical Navigatrics* is not appropriate as a full sequencer. It offers a unique interface for controlling major sequencing events, along with an intuitive means for recording effects. It works well for flowing events, not detailed sequence work. While certainly an interface allowing much more advanced sequencing could be worked out, there is a point at which the tags need be no more than buttons, and thus the tagging interface would offer nothing special.

However, it is in the ability to provide a practical intuitive interface that its strength lays. The interest in this interface lies primarily in exploring the union between sequencing and expressive performance. It is able to act as an interesting continuous MIDI controller interface, but still have sensible discrete actions that can be used for controlling larger-scale events. In comparison to the two commercial alternate interfaces that were discussed earlier, the Korg adds the standard buttons to its *Kaos Pad* in order to control large-scale events, and while it would be possible to map locations on the pad to specific sequencing events, there would remain no clear delineation between choosing effects parameters and sequence parameters. The *airFX* is able to offer even less.

The intuitive, highly-adaptable interface of *Musical Navigatrics* enables it to offer a wide range of capabilities. *Musical Navigatrics* is able to provide a fair amount of flexibility in the way it records. The fact that practically every combination of movement is recordable is relatively

unusual. It is possible to record multiple effects upon one or two lines simultaneously in an intuitive real-time interaction. Although this feature is certainly not unique, it is atypical of device that is not specifically designed as a high level sequencer. Part of why this is worth implementing with *Musical Navigatrics* is that the interface is well suited to make sense of such actions, both on the hardware side and the user side.

In the end, *Musical Navigatrics* lends itself well to basic sequencing. What it offers in real-time capability and easily discernable states makes it sensible for live performance and creative work. The present sequencing software achieves this to some degree but needs expansion. The basic sequencing abilities need to be significantly extended. Along with this, the existing actual sequencing capability needs to be made somewhat more accurate and brought more into line with standard MIDI file format. Timing is not quite consistent in the present implementation and using the *MAX seq* command results in *Musical Navigatrics* requiring its own unique file format. As a result, adding new prerecorded sequences from outside *Musical Navigatrics* can be quite difficult. It must either be played into *Musical Navigatrics* where it is re-recorded, or the file must be completely reformatted. This has been significantly improved with the introduction of software version II using the Propellerheads *Reason* software package. In version II, prerecorded sequences such as the drum tracks can exist in *Reason*, making *Musical Navigatrics* MIDI file format compatible and capable of linking to other standard sequencer capabilities.

The last area the sequencer needs is the addition of supporting graphics. Sequencing is much easier with visual cues to indicate which bar is being played, how long the sequence is, what the tempo is, etc. Also, this would provide additional clarity and feedback as to the state of the tagging interaction. Although it is generally possible to tell the specific interactions occurring in *Musical Navigatrics* based on the tags in the sense field, the action of one tag may be changed by

the presence of another, introducing possible confusion. Graphics relating the current software state can alleviate that. With a graphically clear state, it becomes more practical to make the logical addition of mode switching. For instance, adding a switching tag that toggles between a sequencing mode (where tags are used for expanded sequence handling) and a play mode similar to what presently exists would be a powerful improvement, made much more feasible with graphic support. The present visuals however, remain the same as those used by *Musical Trinkets*, and thus have only marginal relation to the tag behaviors seen in *Musical Navigatrics*. Rewriting the graphics to both relate to the new tag functions as well as relay important sequencing information could significantly improve the sequencing ease of use, as well as provide the user with more a interesting and rewarding interaction.

Chapter 6

Future Development

Although much work has already occurred on the swept-frequency tag reader and much has been discussed about the tag reader in relation to musical interfaces, there is still more potential for technical improvements, development of more complex and more powerful interaction through expansion of positioning capabilities, and also applications of tag reader's capabilities outside the narrow field of music. As such, there are still substantial possibilities for improvement and exploration.

6.1 Improvements

There remain a number of improvements, both large and small, that can be done to the present tag reader. Some of them can be easily implemented with the existing board, while others require a major rethinking of the entire circuitry.

Auto-calibration: One of the more difficult user interactions with the tag reader is manually tuning the frequency sweep on the tag board. Tuning requires an oscilloscope and a fair amount of experience with how altering one resistance will effect the sweep and thus the need to alter other resistances. Even with practice, a severely out of tune board can be very complicated and time-consuming to set up. In order to standardize the sweep, remove the bulk of the tricky frequency tuning process, and most importantly free the tag reader from reliance on expensive extra hardware, it would be useful to enable it to auto-calibrate. Pleasantly, this could be done fairly easily with the

existing board setup. As depicted in Figure 3-2 and discussed in Chapter 3, there are three variable resistances that are used to determine the frequency sweep. These resistors control the exponential rate, exponential bias, and exponential gain. Replacing the user-modified resistors with digitally controlled potentiometers or the Cygnal's two DACs would enable the Cygnal to take full control of tuning. Of these resistors, calibration can be made to depend primarily on balancing the exponential bias vs. the exponential rate while leaving the exponential gain at a preset value. With the installation of dynamic frequency drift stabilization, the Cygnal now has control over both the needed resistances (the control of exponential bias has already been discussed). Control over the exponential rate was enabled by similarly adding an AD5220 in series with the resistance adjusting RC voltage decay time constant.. By counting the wave cycles during a preset amount of time at the beginning and end of the sweep, it should be feasible to develop an algorithm that adjusts the exponential rate and bias to ensure that all tags are reached in the sweep, and that hand calibration is no longer necessary. Although the present board is not set up for it, it would also be possible to dynamically set the output DC baseline level to the minimum possible value thus optimizing dynamic range. At present, the baseline is chosen visually and manually adjusted. By using the Cygnal, it could be automatically set based on the actual sampled data.

Other Improvements: Auto-calibration would be probably the easiest yet also most useful upgrade to the tag reader at the moment. Beyond that, there are more drastic revisions. First, the possibility of upgrading the final stages of the output drive have already been investigated. An alternative to the present push-pull power transistor stage has been designed by Responsive Environments RA Mark Feldmeier. The new driver smoothes irregularities in the final output drive

to the bridge, while also allowing additional improvements in the signal-to-noise ratio. This drive does need further testing but is so far promising.

Another area that can be improved is in linearization of the received analog output. Due to the nature of the solenoidal magnetic field, the amount of coupling between the search coil and the a tag decreases in an exponential manner as the tag moves further from the coil. The result is that rather than a straightforward linear interaction, the implied velocity of a tag changes quickly when close to the search coil and slowly when further away. Adding either hardware or software to make the tag reader output change in a more consistent manner would aid in simplifying user interaction.

Finally, more work needs to be done to discover a means for increasing the spatial range of the tag reader and increasing the number identifiable tags. Some of this can be achieved through more careful selection of tags and components, but some of this requires more drastic changes. At this stage, it is appropriate to look at other means of tag detection. The inductor bridge is definitely successful, but is a somewhat brute force solution, requiring a lot of wasted power and introducing extra tuning issues. It would be promising to explore new methods to detect a tag's impact on the search coil.

6.2 Expansion- Multi-Coil Geometries

One of the major regions for development and expansion is in the use of multi-axis coil geometries to provide tracking information in multiple dimensions, as well as separating orientational information from distance information. Multi-coil geometries can also reshape the magnetic field when driven simultaneously. At present, the Responsive Environments Group's expansion into multi-coil geometries has only begun to explore the capabilities of the tag reader as a multi-dimensional tool.

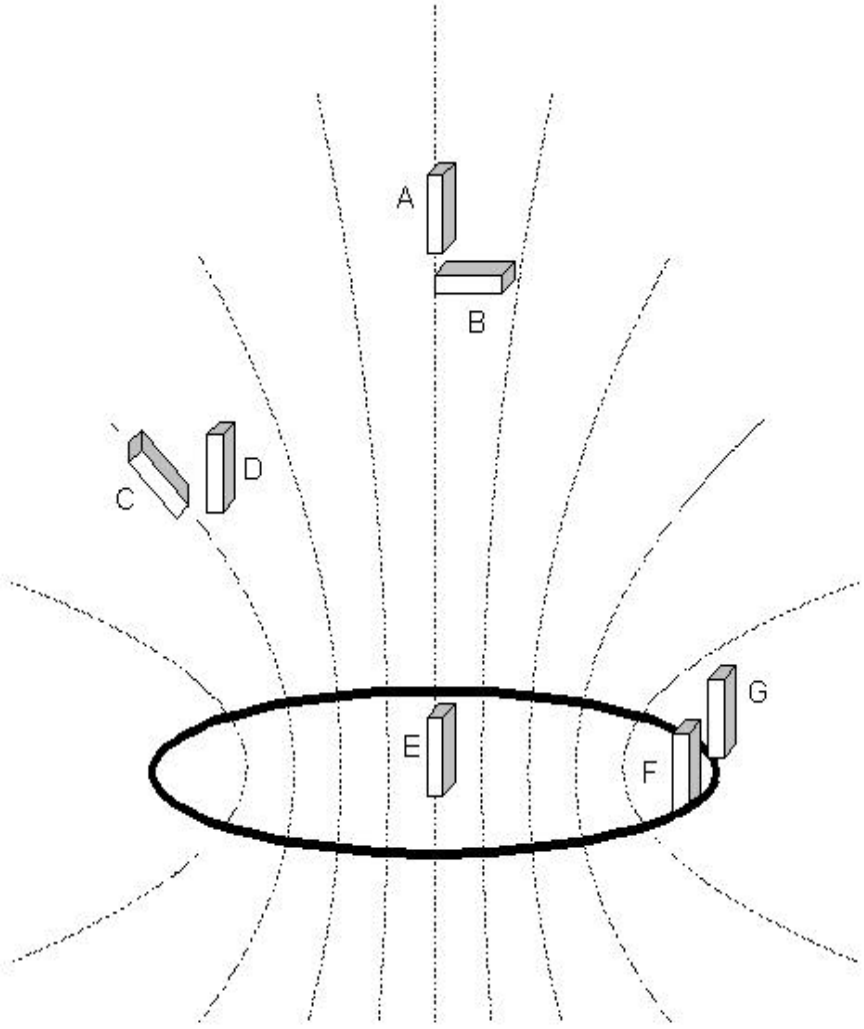


Figure 6-1: Magentic Field Lines and Non-Uniform Interaction
This figure explores the unexpected interactions that occur due to the non-uniform magnetic field in the plane above the search coil. First, while Tag A will be detected by the coil, Tag B will not as Tag A is perpedicular to the magnetic field while Tag B is not. The situation is the same with Tags C and D except here the field lines have bent so that maximum coupling is no longer is achieved through verticle positioning. Here, Tag C will couple more strongly than Tag D even though Tag D is perpendicular to the coil. In the case of Tags E and F, Tag F will couple more-strongly with the search coil as it is in closer proximity to the coil despite the fact that Tag E is in the middle. Lastly, even though Tag G is just as physically close to the coil as Tag F, tag

G will fail to couple at all as it is not perpendicular to the field lines which at the edge of the coil, are mostly horizontal.

A drawback of all the single coil applications to date is that the magnetic field generated by the solenoid coil does not behave in a flat, planar manner. Instead, the magnetic field is curved, and the strength is strongly related to the distance from the perimeter of the actual coil, not from the coil as a whole. This curvature of the field is confusing to a player. As depicted in Figure 6-1, in order to detect a tag right above the actual wire of the coil (Tag G), it must not be perpendicular to the plane of the coil. Whereas, if it were orthogonal to the plane of the coil at the coil's center (Tag E), it would couple very strongly. The typical intuitive user interaction would expect the signal strength to be consistent with respect to orientation, regardless of where the tag is in relation to the coil.

Building on this is the more obvious result that even if the magnetic field were entirely perpendicular to the coil, orientation would still dramatically affect signal strength. This concept is fairly easy for the user to grasp and can be used to produce some interesting spatial mappings, but it is generally not managed well by the single-coil reader's tag interaction. The tag reader is unable to distinguish whether a singly-tagged object is being rotated or being moved closer to the coil. Again, orientation sensitivity can provide for useful interaction, for instance the rotational interaction with the triple-tag objects, but when combined with other non-uniformities, orientation dependence primarily adds undesired complexity.

A third unintuitive non-uniformity is that the magnetic field generated by the current through the read coil's wound wire is strongest near the wire. This situation is depicted using Tags E and F in Figure 6-1, and results in the tag reader interpreting translational movement across the coil plane the same as it would if the tag were moved up or down.

The end result and combination of these three non-uniformities is that the actual dynamics of the playing field are, in fact, much more complex than simple proximity. These dynamics can be understood and made useful to the user, but on the whole result in the tagging interface being much harder and less intuitive than expected. Adding more coils can help alleviate and make better use of these non-uniformities.

By driving two coils concurrently, the magnetic field lines can be significantly reshaped. The most straightforward geometry produced by two coils is with the Helmholtz coil configuration. As depicted in Figure 6-2, by placing two coils parallel to each other and appropriately spaced, the magnetic field in the area between the two coils will become significantly more uniform and constant. Although the addition of an opposing coil significantly alters the physicality of the interactive space, it also removes any complexity in the shape of the magnetic field lines, resulting in an easier to understand coupling strength characteristics. Additionally, two opposing coils allow for the orientation of a tag to be easily distinguished from the location of the tag. This is due to the fact that the overall tag coupling sensed by both coils will indicate orientation while the relation of one coil's response to the other will imply overall distance between the two.

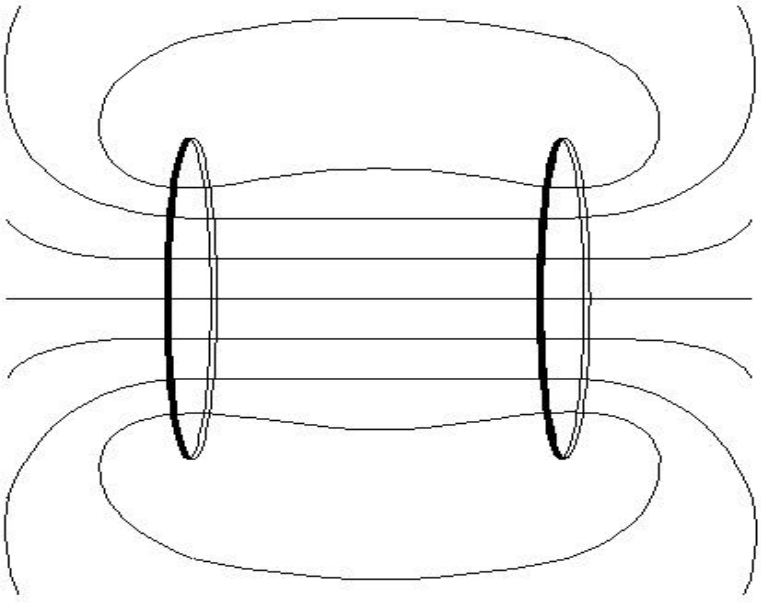


Figure 6-2: Magnetic Field Lines from a Helmholtz Configuration

As discussed by Hsiao[1], a rudimentary 6 coil cubic tag reader has been developed and can provide reasonable 3-dimensional tracking within a 1-2' cube. The cube tracker is built by using three pairs of Helmholtz coils, one for each pair of faces. As previously discussed, each pair of faces is run individually, so that only one axis is being used at a time. Despite some issues, the cube tracker provides a significant step up from the single Heimholz configuration as it can detect location and orientation on each of the three axes. This information can be adapted for variety of 3-dimensional tracking applications. For a more complete discussion of the 3-dimensional cube tracker, including problems, successes, and the test application associated with it, refer to Hsiao's thesis[1].

Aside from using Helmholz coils, there remain a number of interesting multi-coil geometries. One of the problematic interactions that has been evidenced by users of *Musical Trinkets* and *Musical Navigatrics* is the lack of any significant response to transverse motion. It is very common to see a user want to use the tags by dragging an object across the sense table rather

moving it up and down over the table. This action produces little response with the current single coil system. A simple multi-coil geometry that, while not providing the same benefits as a Helmholtz coil, does provide some translational information would be highly beneficial.

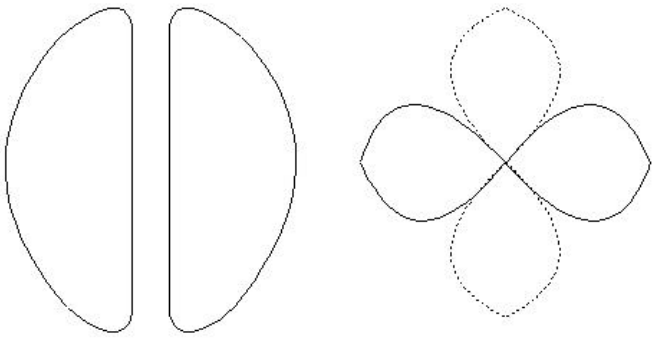


Figure 6-3: Exploratory Coil Geometries

Two dual-coil, synchronous-drive geometries presently under consideration are depicted in Figure 6-3. These would both provide for definite translational response. The figure on the left is good for left-right information while the figure-eight geometry on the right is particularly interesting in that it should produce an effect similar to an absolute value joystick, where moving a tag in a positive direction on the X-axis will be appear the same as moving it in the negative direction, but the magnitude of the change should be detectable. This behavior should also occur for the Y-axis so that the position of a tag in any XY quadrant is determinable even though the actual quadrant the tag is in is not. No matter how a tag is moved in the plane, a response will be produced.

Chapter 7

Conclusions

The need for alternative interesting but intuitive and easy to use interfaces is a reality that will surely grow in the future as digital interaction continues to permeate daily life. Although alternate means of human-computer interface are being explored on all fronts, nowhere is the need for an alternate interface so apparent and also apt as in musical controllers. The capability of digital synthesis and the recent commercial appearance and rapid success of such devices at least within the growing dance music community clearly outlines the need for an expressive musical interface. Passive tagging offers a unique and appealing possibility for developing these new interfaces.

This project began with a previously developed continuous swept RF tag reader that was capable of simultaneously detecting the continuous position of up to 20 distinct tags in real-time within a roughly 12" distance from the search coil. The new board developed under this research improves the sensitivity and dynamic range of the previous tag reader through the addition of a multiply sampled baseline as well as faster and more sensitive sampling. Along with this, the problem of stability, a problem in the previous design, has been largely solved by the addition of a digitally controlled potentiometer capable of reliably counteracting dramatic frequency shifts. Any additional drift not taken care of by the potentiometer can also be monitored through the use of four known calibration tags spaced throughout the 50kHz-400kHz sweep. Moreover, the new tag reader should be eminently and imminently capable of handling complete auto-calibration. Additionally,

the new board has been specifically designed for the development of multi-coil geometries. The tag reader can now run up to 3 asynchronous read coils (hence sensitive axes) on one board.

Besides advances in the capabilities of the tag reader, the development of the swept tagging interface as a complex free gesture musical interface has been significantly advanced and explored. The implementation of a new application, *Musical Navigatrics*, has introduced significant new depth and potential to what was previously little more than a musical toy. *Musical Navigatrics* has successfully demonstrated the outstanding nature of the tagging interface to implement an exciting and needed means for effects control. *Musical Navigatrics* has illuminated the expressivity, flexibility, and ease of use for controlling effects inherent in the tagging interface.

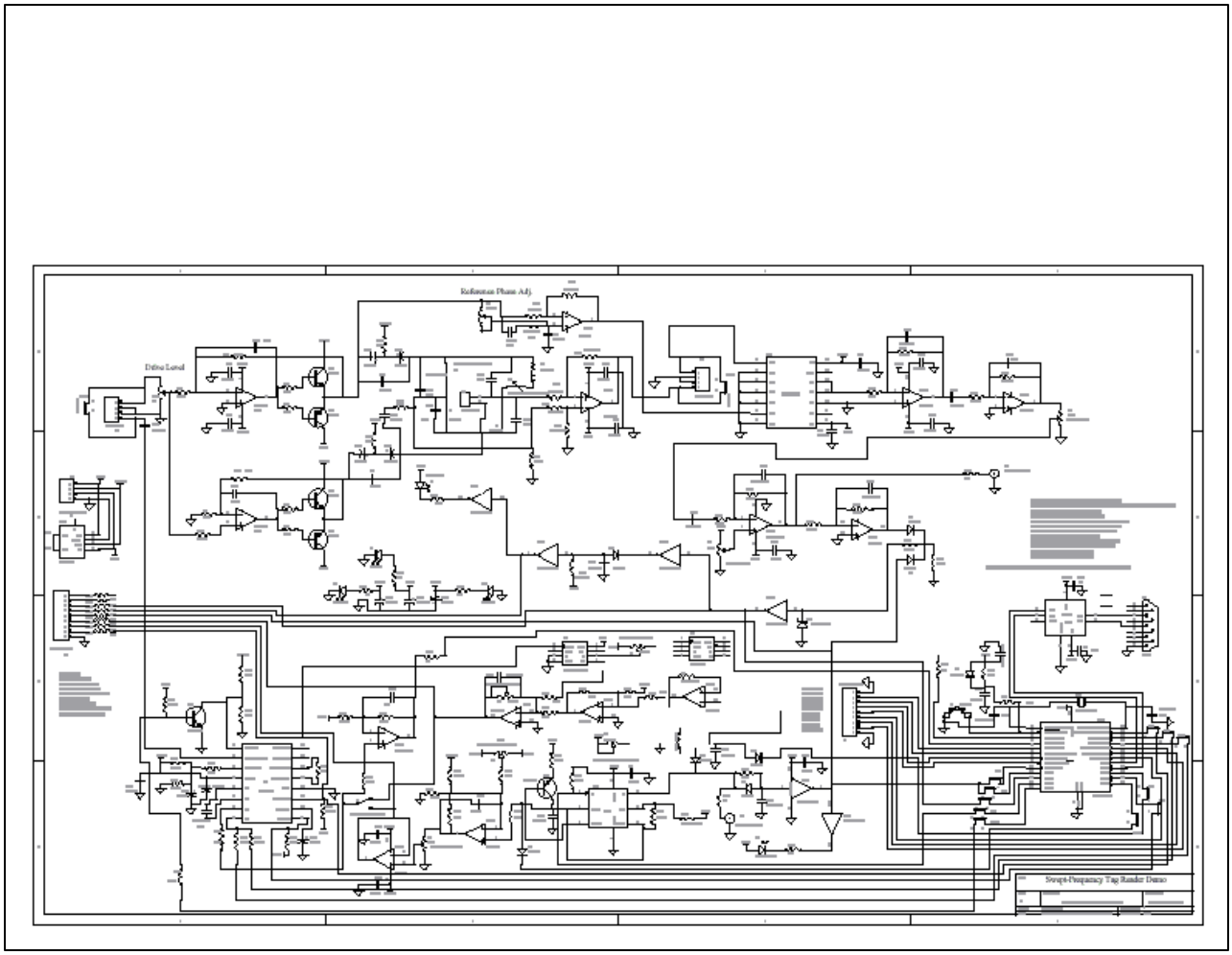
At the same time, *Musical Navigatrics* has exposed problems in using the tag reader as a pitched MIDI instrument. Despite promise, the tagging interface has difficulty surviving the typical problems associated with free gesture instruments. It suffers from a lack of direct feedback, as well as the need for more complex control despite the limited number of hands available for playing. On the positive side, it shows outstanding potential for high-level tonal shaping (e.g. a curve generator or arpeggiation controller).

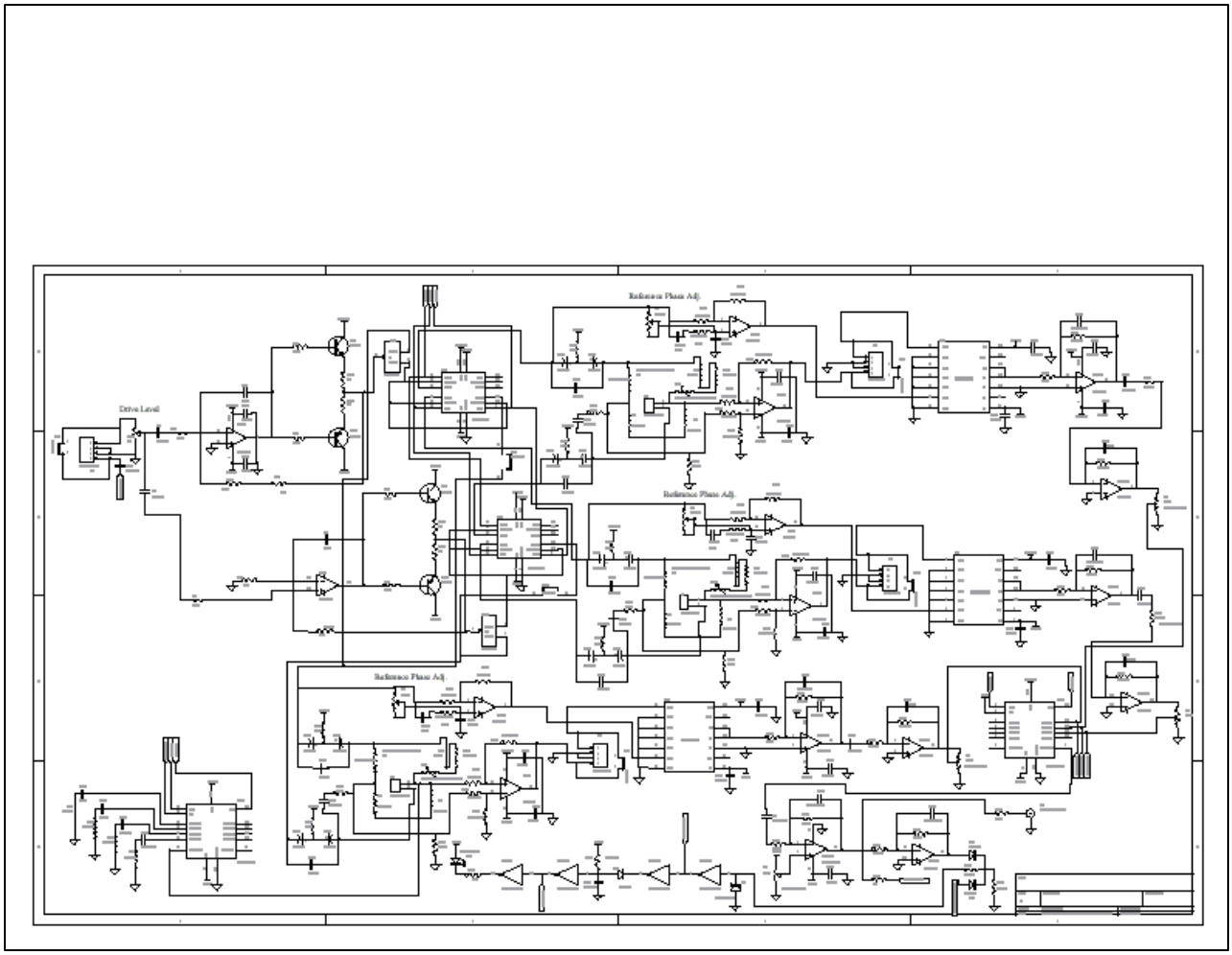
Lastly, *Musical Navigatrics* has demonstrated some of the possibilities for a tagging interface as a complete performance or creative platform. The tag reader's unique ability to understand and provide intuitive control of both discrete and continuous events presents a useful balance between expressivity and sequencing. Although still in need of substantial software development both musically and graphically, *Musical Navigatrics* manages to present a fun, interesting, and potentially powerful means for music composition and expressive live electronic performance.

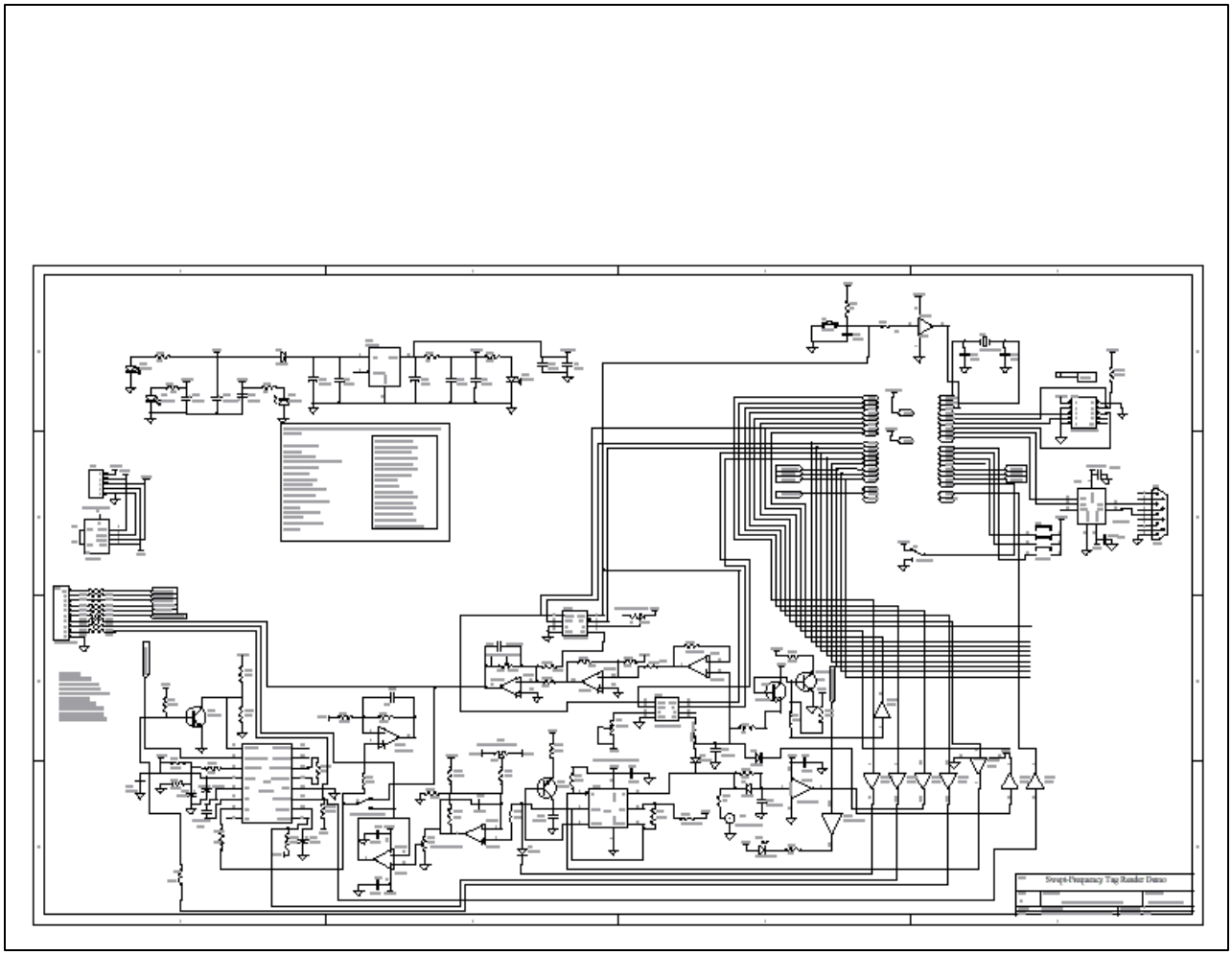
Beyond *Musical Navigatrics* is the need for more fully investigating and implementing the tagging interface with multi-coil geometries. Multi-coil geometries not only enable different magnetic configurations that can improve the uniformity of detected coordinates but also add more physical dimensions to the otherwise one-dimensional tagging interface. Besides further development of a cube tracker, simple two-coil arrangements should be explored to break special degeneracy, providing users with a more satisfying interaction while adding only a minimum of technical complexity.

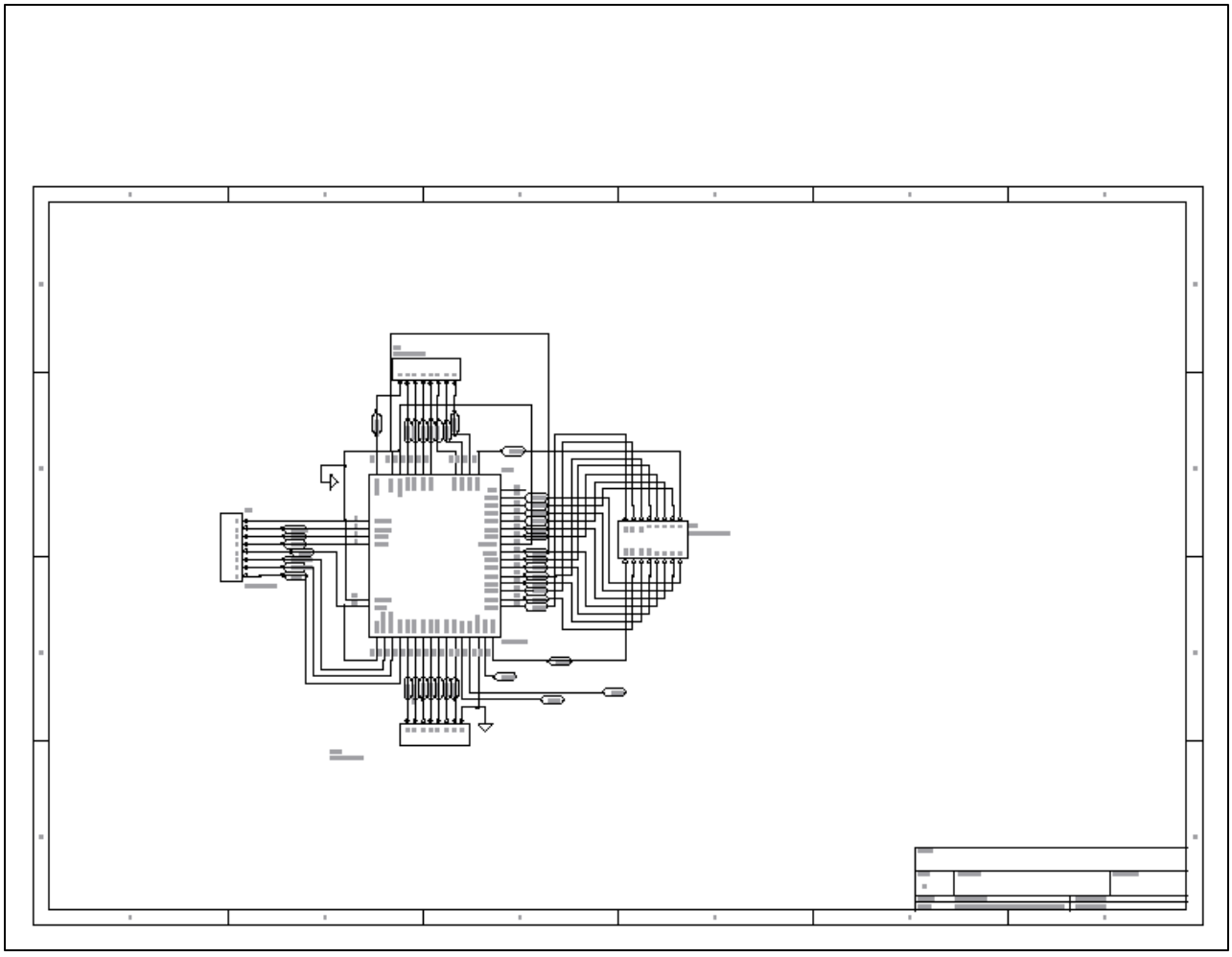
Appendix A: Schematics

This appendix presents the schematics for the original tag reader and the drive circuit, oscillator circuit, and control circuit of the new reader.




Reference Phase Adj.
Drive Level
Reference Phase Adj.
Reference Phase Adj.


Swept-Frequency Tag Reader Demo



Appendix B: Code

This appendix provides the code for the Cygnal C8051F0005 microprocessor. Includes the header file cygtag.h and the executing file, cygtag.m.

B.A Cygtag.h: Microprocessor Header Selections

Cygtag.h is composed of the standard Cygnal 8051F0005 header file plus the following code:

```
/*-----------------------------------------------------------------------
;          Copyright (C) 2000 CYGNAL INTEGRATED PRODUCTS, INC.
;          All rights reserved.
;
;
;          FILE NAME       : C8051F000.h
;          TARGET MCU      : C8051Fxxx (C8051 System Controller)
;          DESCRIPTION     : Register/bit definitions for the C8051Fxxx family.
;
;          REVISION 1.8
;-----------------------------------------------------------------------*/

/* Pin labels */

sbit PIN_A0 = P0^0;
sbit PIN_A1 = P0^1;
sbit PIN_A2 = P0^2;
sbit PIN_A3 = P0^3;
sbit PIN_A4 = P0^4;
sbit PIN_A5 = P0^5;
sbit PIN_A6 = P0^6;
sbit PIN_A7 = P0^7;

sbit PIN_B0 = P1^0;
sbit PIN_B1 = P1^1;
sbit PIN_B2 = P1^2;
sbit PIN_B3 = P1^3;
sbit PIN_B4 = P1^4;
sbit PIN_B5 = P1^5;
sbit PIN_B6 = P1^6;
sbit PIN_B7 = P1^7;

sbit PIN_C0 = P2^0;
sbit PIN_C1 = P2^1;
sbit PIN_C2 = P2^2;
sbit PIN_C3 = P2^3;
sbit PIN_C4 = P2^4;
sbit PIN_C5 = P2^5;
sbit PIN_C6 = P2^6;
sbit PIN_C7 = P2^7;

sbit PIN_D0 = P3^0;
sbit PIN_D1 = P3^1;
sbit PIN_D2 = P3^2;
sbit PIN_D3 = P3^3;
sbit PIN_D4 = P3^4;
sbit PIN_D5 = P3^5;
sbit PIN_D6 = P3^6;
```

```
sbit PIN_D7 = P3^7;

// standard bools
#define false 0
#define true  1

// TIMER 0 and 1 Settings
#define COUNTER13       000
#define TIMER13         100
#define COUNTER16       001
#define TIMER16         101
#define COUNTER8        010
#define TIMER8          110
#define OTHER           111

// ADC gain values

#define GAIN1   0
#define GAIN2   1
#define GAIN4   2
#define GAIN8   3
#define GAIN16  4
#define GAIN.5 7
```

B.B Cygtagm.c: Microprocessor Code

/* Code to run Cygnal C8051F0005 for swept-frequency tag reader. Starts with general board initialization. At this point, board can be held for hand calibration (Calibration Toggle Switch) prior to sampling and // storing of baseline (for upto 3 axes). This is followed by a sweep of the test tags in order to figure out the point where each tag should be switched in. This completes calibration and initialization.

At this point, the board goes into a loop whose prime function is to take samples of the frequency sweep. The length of each sweep is also checked for frequency drift and may trigger pot. to hold sweep length. The calibration/test tags will also be switched in every specified number of sweeps.

This particular version of code is running all three axes*/

```
#include "stdio.h"
#include "math.h"
#include "intrins.h"
#include "cygtag.h"

#define PIN_TRIGGER PIN_B1
#define PIN_INPUT 0
#define PIN_CAL 1
#define PIN_TAGPRESENT PIN_B2
#define PIN_RESETSWEEP PIN_B3
#define PIN_OSCILLATE PIN_B4
#define PIN_EXPOTRIG PIN_B5
#define PIN_HI555 PIN_B7
//make this back to 6 for board
#define PIN_RUNCAL PIN_B0   //changed from b6 to free up LED

/* alternate for board circuit  (test board are odd even grouped) */
#define PIN_MUX0 PIN_C0             //these are happy at 3.3V
#define PIN_MUX1 PIN_C1
#define PIN_MUX_EN PIN_C2

//give rate 4,5
#define PIN_BRES_EN PIN_C4          // Exponetial Bias Res
#define PIN_BRESDIR PIN_C5
#define PIN_RES_EN PIN_C6           //these are happy at 3.3V
#define PIN_RESDIR PIN_C7           // Exponetial Rate Res
#define PIN_RUNSAMPS PIN_D0

// AXIS specific pins
#define PIN_AXISM0 PIN_D1
#define PIN_AXISM1 PIN_D2
#define PIN_AXISM2 PIN_D3

/*
#define PIN_MUX0 PIN_C3             //these are happy at 3.3V
#define PIN_MUX1 PIN_C7
#define PIN_MUX_EN PIN_C1

#define PIN_RES_EN PIN_C0           //these are happy at 3.3V
```

```
#define PIN_RESDIR PIN_C4
#define PIN_RUNSAMPS PIN_D0

// AXIS specific pins
#define PIN_AXISM0 PIN_D1
#define PIN_AXISM1 PIN_D3
*/

#define XTLVLD_BIT      0x80        // OSCXCN.7 Crystal osc valid flag
//#define   LED PIN_B6              // green LED: '1' = ON; '0' = OFF

#define CALIB_RUNS 300              // how many calibration runs done per axis
#define MAXSAMP 900                 // number of samples taken
#define SAMP_SPEED 64430;

void clocksConfig (void);
void xbarConfig (void);
void uartConfig (void);
void pcaConfig (void);
void setTimer3 (unsigned int val);
//void setTimer1 (unsigned int mode, unsigned int val);
void adcConfig (unsigned int gain);
void setAdcChannel (unsigned int val);
unsigned int readAdc (void);
void usDelay (unsigned int val);

void ttag1On (void);
void ttag2On (void);
void ttag3On (void);
void ttag4On (void);

void addbuf(char val);
char getbuf(void);
//void addbuf(unsigned int val);
//unsigned int getbuf(void);
void driftTest (void);
void loop (void);

unsigned int tmpSmp = 0;
unsigned int i, j;
unsigned int overflows=0;
unsigned int threshold;
unsigned int counter=0;
data unsigned int last, start;
data char axis=0; // which pair of boards are we running now?  control the mux
long sum;

unsigned int ttag2, ttag3;
unsigned int ttag4 = 0xffff;
data unsigned int bufstart=0, bufend=0;
idata char buffer[80];

                  //AXIS
char even = true;
char calib = false;

unsigned int axisOffset;  // *AXIS*
```

```
data unsigned int basePo = 0;
unsigned int baseBuf[MAXSAMP];       // 289 samples should cover the current
sample-sweep rate


//unsigned recal_count=0; // two bytes, hopefully; count cycles 'til
reclibration
//char linear=1;

void main (void)
{
   unsigned int last = 0;
   unsigned int pres = 0;
   unsigned int next = 0;
   unsigned int high = 0;
   data unsigned int pca = 0;
   long int countSum = 0;
   unsigned int sweeps = 0;
   unsigned int tmpStart, tmpEnd, tmp;
   unsigned int tagMag, tagMax, tagCont, tagStart, tagEnd;
   unsigned int freq_count = 0;
   unsigned int sampRate = SAMP_SPEED;


//   unsigned int bufTmp[MAXSAMP];


   clocksConfig();
   xbarConfig();
   uartConfig();
   pcaConfig();
   adcConfig(GAIN1);

   printf("PIC starting up...\n");
   putchar (253);
   putchar(255);
   putchar(255);
   threshold = 0;
   PIN_RESETSWEEP = 1;
   PIN_OSCILLATE = 1;
   PIN_EXPOTRIG = 1;
   PIN_RUNCAL = 1;
   PIN_HI555 = 1;
   PIN_MUX_EN = 0;
   PIN_RES_EN = 1;                  // unenabled

// temp

      PIN_AXISM2 = 0;
   j = 0;
//   LED = ~LED;
   setAdcChannel(PIN_INPUT);
   while (basePo < MAXSAMP)         // init baseline
       baseBuf[basePo++] = 0;
   basePo = 0;
 //  usDelay (200);
```

```
   for (i=0; i<79; i++)
     buffer[i]=0x11;


// takes baseline thresholds
// set timer 3 so that sample rate half that during regular detection (baseline
is
// half as accurate
     setTimer3(sampRate);

// won't run unless run switch on (this allows for calibration with
// no resistor compensation
//   while (!(PIN_RUNSAMPS));

// new *AXIS* handling code :
   for (axis=0; axis < 3; axis++) {
     axisOffset = axis * (MAXSAMP / 3);   // this provides offset for where in
buffer should be
       if (axis == 1) PIN_AXISM0 = 1;
         else PIN_AXISM0 = 0;
     if (axis == 2) PIN_AXISM1 = 1;
         else PIN_AXISM1 = 0;
            // *AXIS* end new code

     while (!(PIN_TRIGGER));
     while (PIN_TRIGGER);

     PCA0L = 0;         //reset PCA counter
     PCA0H = 0;

     CR = 1;            // turn on PCA counter
     TMR3CN  = 0x06;    // timer 3 to run off sysclk and turn on


       for (i=0; i<CALIB_RUNS; i++)
       {
        PIN_D6 = 0;                 // just in here to help debug timing
//        usDelay(600);
          while (!(PIN_TRIGGER))
        {
               TMR3CN = 0x06;       // clear overflow flag
               if (even) {
               threshold = readAdc();
               PIN_D4 = ~PIN_D4;
                 basePo++;
                 if (baseBuf[basePo + axisOffset] < threshold)    //*AXIS*
                   baseBuf[basePo + axisOffset] = threshold;
      //*AXIS*
                   even = false;
          }
               else even = true;

               while (TMR3CN != 0x86);
        }
          PIN_D6 = 1;

          TMR3CN &= 0xfb;     // stop sample clock
```

```
          CR = 0;               // stop PCA
//      putchar(threshold >> 8);
//        putchar(threshold&0x00ff);
//        putchar(basePo >> 8 );   // use in conjunction with tag calib to get
#samples
//        putchar(basePo&0x00ff);
          pca = PCA0L;                    // must read low first
          pca += PCA0H << 8;
          putchar(pca >> 8);
          putchar(pca&0x00ff);
          putchar(255);
          putchar(255);

          countSum += pca;    // want average length of sweep

          basePo = 0;
            even = true;
          PCA0H = 0;
          PCA0L = 0;
          TMR3H = 0xff;                   // sample immediately
        TMR3L = 0xff;

        while (PIN_TRIGGER);
          CR = 1;
          TMR3CN |= 0x04;   // turn clock back on;
      }
   } //*AXIS*

   freq_count = countSum / (CALIB_RUNS * 3);   //*AXIS*

   putchar(freq_count >> 8);
   putchar(freq_count&0x00ff);
   putchar(252);
   putchar(254);

//  This shouldn't be necessary, if anything, tend to get one extra sample at
the
//  threshold (takes 578 samples (at 2 times over samp), but just in case...
// basePo = MAXSAMP - 10
/*   while (basePo < MAXSAMP)
      if (baseBuf[basePo++] < threshold)
        baseBuf[basePo] = threshold;   // ensure that can accomodate extra
samples
*/

   basePo = 0;
   while (basePo < MAXSAMP)
   {
    basePo++;
      if (baseBuf[basePo] > 4045)
        baseBuf[basePo] = 4095;  //add saftey margin
      else baseBuf[basePo] = baseBuf[basePo]+60;
   }
/*
   for (basePo = 1; basePo <= MAXSAMP; basePo++)
     baseBuf[basePo] = bufTmp[basePo];
*/
```

```
   last = baseBuf[1];
   pres = baseBuf[2];
   next = baseBuf[3];

// for any sample, want the highest on each side
   if (last > pres)
     baseBuf[1] = last;       // fill in first sample
   else
       baseBuf[1] = pres;
   for (basePo = 2; basePo < MAXSAMP; basePo++)
   {
       high = last;
       if (pres > high )
            high = pres;
       if (next > high)
          high = next;
       baseBuf[basePo] = high;
       last = pres;
       pres = next;
       next = baseBuf[basePo + 1];
   }
   if (next > pres)
      baseBuf[MAXSAMP] = next;
   else baseBuf[MAXSAMP] = pres;

   basePo = 0;

// take samples from drift tags (using Timer 1)
//   these are for computer use so just send and don't worry
    while (bufstart!=bufend)              // empty buffer
         {
            putchar(getbuf());
         }


   ttag1On();           // start with tag 1 on
   PIN_MUX_EN = 1;

   tagMax = 0;          //          inits
   tagEnd = 0;
   tagStart = 0xFEFE;
   PCA0H = 0;
   TMR3H = 0xff;                    // no delay
   TMR3L = 0xff;


   while (!(PIN_TRIGGER));          // reset start timing
   while (PIN_TRIGGER);

   calib = true;
   EIE2   |= 0x01;   // enble Timer 3 overflow interrupt
   CR = 1;               // turn on PCA counter
   TMR3CN  = 0x06;   // timer 3 to run off sysclk and turn on
   EA = 1;

// skip all this stuff if you are running multi-axis, or just do it on
```

```
// one axis.

   for (i = 0; i < 200; i++)
      {
      while (!(PIN_TRIGGER));
        PIN_D6 = 1;
        PIN_D4 = 0;
        EA = 0;                     // disable interrupts
        CR = 0;
        TMR3CN &= 0xfb;             // turn off clock

/*      pca = PCA0L;
        putchar(PCA0H);
        putchar(pca);
        putchar(255);
        putchar(255);*/

        basePo = 0;
        even = true;
        PCA0L = 0;
        PCA0H = 0;
      TMR3H = 0xff;                         // no delay
      TMR3L = 0xff;

        addbuf(255);                        // mark end of new data
        addbuf(255);

        tmpStart = (getbuf() << 8);
        tmpStart += 0x00ff&getbuf();
        getbuf();                           //this is to pull out the debug 255

        if (tmpStart != 65535)              // did we get any info?
            {
              tmpEnd = (getbuf() << 8);
              tmpEnd += 0x00ff&getbuf();
              getbuf();                           // this is to pull out debug
"255"

              tagMax = (getbuf() << 8);
              tagMax += 0x00ff&getbuf();

              tagCont = (getbuf() << 8);
              tagCont += 0x00ff&getbuf();
              getbuf();       // debug

              while (tagCont != 65535)    // while still getting tag info
              {
                  tmpEnd = (getbuf() << 8);
                  tmpEnd += 0x00ff&getbuf();
                  getbuf();         //debug

                  tagMag = (getbuf() << 8);
                  tagMag += 0x00ff&getbuf();
                  if (tagMag > tagMax)
                        tagMax = tagMag;

                  tagCont = (getbuf() << 8);
```

```
            tagCont += 0x00ff&getbuf();
            getbuf();         //debug
          }
      }

if (tmpStart < tagStart) tagStart = tmpStart;
if (tmpEnd > tagEnd) tagEnd = tmpEnd;

switch (i)
{
      case 50: {
            putchar(tagStart >> 8);
            putchar(tagStart & 0x00ff);
            putchar(tagEnd >> 8);
            putchar(tagEnd & 0x00ff);
            putchar(tagMax >> 8);
            putchar(tagMax & 0x00ff);
            putchar(255);
            putchar(254);

            ttag2 = tagEnd;
            tagEnd = 0;
            tagStart = 0xFEFE;
            tagMax = 0;

            ttag2On();
            break;
      }                 // This tag no good right now

      case 100: {
            putchar(tagStart >> 8);
            putchar(tagStart & 0x00ff);
            putchar(tagEnd >> 8);
            putchar(tagEnd & 0x00ff);
            putchar(tagMax >> 8);
            putchar(tagMax & 0x00ff);
            putchar(255);
            putchar(254);

            if (tagStart < ttag2) ttag2 = tagStart;
            ttag3 = tagEnd;
            tagEnd = 0;
            tagStart = 0xFEFE;
            tagMax = 0;

            ttag3On();
            break;
      }
      case 150: {
            putchar(tagStart >> 8);
            putchar(tagStart & 0x00ff);
            putchar(tagEnd >> 8);
            putchar(tagEnd & 0x00ff);
            putchar(tagMax >> 8);
            putchar(tagMax & 0x00ff);
            putchar(255);
            putchar(254);
```

```
                  if (tagStart < ttag3) ttag3 = tagStart;
                  ttag4 = tagEnd;
                  tagEnd = 0;
                  tagStart = 0xFEFE;
                  tagMax = 0;

                  ttag4On();
                  break;
            }
            default:
                  break;
      }
      while (PIN_TRIGGER)  // if there is a tag present, this takes about
      {
//          _nop_;
        }

       CR = 1;
     TMR3CN  = 0x06;          // timer 3 to run off sysclk and turn on
       PIN_D6 = 0;
     EA = 1;                  // start of sweep; take data again

   }

      EA = 0;
    TMR3CN &= 0xfb;           // turn off clock

      putchar(tagStart >> 8);       // put tag4 info on stack
      putchar(tagStart & 0x00ff);
      putchar(tagEnd >> 8);
      putchar(tagEnd & 0x00ff);
      putchar(tagMax >> 8);
      putchar(tagMax & 0x00ff);
      putchar(255);
      putchar(254);

      putchar(ttag2 >> 8);
      putchar(ttag2&0x00ff);
      putchar(ttag3 >> 8);
      putchar(ttag3&0xff);
      putchar(ttag4>>8);
      putchar(ttag4&0x00ff);
      putchar(255);
      putchar(254);

      PIN_MUX_EN = 0;                     // Turn off mux
      calib = false;
      basePo = 0;
      even = true;

// this is just in here as a visible sign that initialization is complete
   for (i = 0; i <100; i++)
     putchar (111);

// won't run unless run switch on (this allows for calibration with
// no resistor compensation  (this time for debug)
```

```
   while (!(PIN_RUNSAMPS));

// with 20mhz clock, we have 200ns cycles.  takes roughly 550 cycles per sample
so
// clock counts should give us 1000-cycle timing, or about 50 us sample rate.

   CR = 0;              // turn off PCA and reset
   PCA0L = 0;
   PCA0H = 0;
   TMR3H = 0xff;                    // no delay
   TMR3L = 0xff;

   setTimer3(sampRate);
   EIE2   |= 0x01;   // enble Timer 3 overflow interrupt


   while (!(PIN_TRIGGER));
   while (PIN_TRIGGER); // should start at beginning of sweep
   CR = 1;               // turn on PCA
   TMR3CN  = 0x06;   // timer 3 to run off sysclk and turn on
   EA    = 1;            // gobal interrupt enable

// loop that runs during sampling sweep and sends results
   while(1)
   {
//      setAdcChannel(PIN_INPUT);
      while (!(PIN_TRIGGER));
        PIN_D6 = 1;
        PIN_D4 = 0;
        EA = 0;                     // disable interrupts
        TMR3CN &= 0xfb;       // turn off clock
        CR = 0;                     // turn off PCA
        PIN_MUX_EN = 0;       // disenable test tags

//      addbuf(overflows);
//      addbuf(counter);
        pca = PCA0L;
        pca += (PCA0H << 8);
        addbuf(pca >> 8);
        addbuf(pca&0x00ff);
//      addbuf(overs);
        addbuf(255 - axis);
        addbuf(255 - calib);
//      addbuf(255 - calib);  // indicate if it is a calibration run
//      addbuf(255 - axis);

// new *AXIS* handling code :
     axis ++;
       if (axis > 2) axis = 0;
     axisOffset = axis * (MAXSAMP/3); // this provides offset for where in
buffer should be
       if (axis == 1) PIN_AXISM0 = 1;
         else PIN_AXISM0 = 0;
     if (axis == 2) PIN_AXISM1 = 1;
         else PIN_AXISM1 = 0;
            // *AXIS* end new code
```

```
        if (pca > (freq_count + 20)) {
             PIN_RESDIR = 1;  // down
             PIN_RES_EN = 0;
        }
        else if(pca < (freq_count - 20)) {
             PIN_RESDIR = 0;  // up
             PIN_RES_EN = 0;
        }
        else
            PIN_RES_EN = 1;

        if (sweeps > 450)
          {
            calib = true;
              ttag1On();
              PIN_MUX_EN = 1;             // add test resonances into sweep
              sweeps = 0;
          }
      else
            calib = false;

//      counter=0;
//      overflows=0;
        basePo = 0;
        even = true;
        sweeps++;

        PCA0L = 0;
        PCA0H = 0;
        TMR3H = 0xff;                     // no delay
      TMR3L = 0xff;

//     PIN_D6 = 0;

      while (PIN_TRIGGER)    // if there is a tag present, this takes about
      {
         if (bufstart!=bufend)
         {
//                getbuf();
            putchar(getbuf());
         }
//          PIN_D6 != PIN_D6;
      }

       PIN_D6 = 0;

       TMR3CN = 0x06;                     // restart clocks
       CR = 1;
       EA = 1;                            // start of sweep; take data again


   }
}

void ttag1On (void)
{
      PIN_MUX0 = 0;
```

```
      PIN_MUX1 = 0;
}

void ttag2On (void)
{
      PIN_MUX0 = 1;
      PIN_MUX1 = 0;
}

void ttag3On (void)
{
      PIN_MUX0 = 0;
      PIN_MUX1 = 1;
}

void ttag4On (void)
{

      PIN_MUX0 = 1;
      PIN_MUX1 = 1;
}


void clocksConfig (void)
{
      // disable watchdog timer
      WDTCN = 0xde;
      WDTCN = 0xad;


      // Start up the external oscillator
      OSCXCN = 0x67;                // enable crystal osc div 1
                                    //  for 20MHz crystal (power ratio not
                                    //  really known)

      // wait for xtal osc to start up
      while ((OSCXCN & XTLVLD_BIT) == 0 )
      {
      }

      OSCICN = 0x88;                            // select external osc as
                                                //  system clock, disable
                                                //  internal osc
}

void xbarConfig (void)
{
      // configuring crossbar
      XBR0 = 0x44;      // connect UART and PCA0 external trigger
      XBR1 = 0x00;
      XBR2 = 0x40;      // enable crossbar
      PRT0CF = 0xff;  // set all unused pins to push-pull
      PRT1CF = 0xf8;       // some are read
      PRT2CF = 0xff;
      PRT3CF = 0xfe;
}
```

```
// Configure the UART using Timer2, for 19200.2kbps, 8-N-1 using 20MHz sysclk
void uartConfig (void)
{
      SCON   = 0x50;                                          // SCON: mode 1, 8-
bit UART, enable RX
      RCAP2H = 0xFF;
//                                  22.11Mhz
      RCAP2L = 0xfa;                                          // 115200 baud
//                                  20Mhz
//    RCAP2L = 0xfc;                                          // 115200 baud
//    RCAP2L = 0xf5;                                          // 57600 baud
//    RCAP2L = 0xdf;                                          // 19200 baud
      RCLK   = 1;
      TCLK   = 1;
      TR2    = 1;
      PCON  |= 0x80;                                          // SMOD = 1

      TI     = 1;                                             // Indicate TX
ready
}

void pcaConfig (void)
{
      PCA0MD = 0x06;    // set PCA0 to count off ECI, disable interrupts
}

void setTimer3 (unsigned int val)
{
   TMR3CN &= 0xfb;            // turn off timer 3

   TMR3CN |= 0x02;            // runs of sys-clock direct
   TMR3RLL = val & 0x00ff;  // set timer 3 to over flow every 1000 clock cycles
   TMR3RLH = val >> 8;
   TMR3H = 0xff;              // init Timer3 to reload immediately
   TMR3L = 0xff;
}

/*void setTimer1 (unsigned int mode, unsigned int val)
{                                        // doesn't deal with counter gate-ing
      TR1 = 0;                // turn off clock prior to setting

      switch(mode){
            case COUNTER13:
                  TMOD = (TMOD&0x0f)|0x40;      //retain timer 0 settings
                  TL1 = val&0x00FF;
                  TH1 = (val >> 8)&0x001F;      // 13 bits only
                  break;
            case TIMER13:
                  TMOD = (TMOD&0x0f)|0x00;      //retain timer 0 settings
                  TL1 = val&0x00FF;
                  TH1 = (val >> 8)&0x001F;      // 13 bits only
                  break;
            case COUNTER16:
                  TMOD = (TMOD&0x0f)|0x50;
                  TL1 = val&0x00FF;
                  TH1 = val >> 8;                     // 16 bits
```

```
                  break;
            case TIMER16:
                  TMOD = (TMOD&0x0f)|0x10;
                  TL1 = val&0x00FF;
                  TH1 = val >> 8;                     // 16 bits
                  break;
            case COUNTER8:
                  TMOD = (TMOD&0x0f)|0x60;
                  TL1 = val&0x00FF;             // 8 bits
                  break;
            case TIMER8:
                  TMOD = (TMOD&0x0f)|0x20;
                  TL1 = val&0x00FF;             // 8 bits
                  break;
            default:
                  // i don't want to do this one, me lazy
                  break;
      }

   CKCON |= 0x10; //  based on CLK
}*/

void adcConfig (unsigned int gain)
{
      AMX0CF = 0x00;    // all inputs are non-differential
      ADC0CF = 0x80+ gain;    // SAR conversion clock set to sysclk/16 so that
SAR < 2Mhz
                        //    as devel. board has Va < 5, temp set gain=0.5
      AMX0SL = 0x00;
      REF0CN = 0x03;                // enable VREF, on-chip bias generator
                                    //  and bias output buffer
      ADCEN = 1;                    // turn on AD //

}

void setAdcChannel (unsigned int val)
{
      ADCEN  = 0;                   // turn off AD
      AMX0SL = 0x01 + val;          // set channel
      ADCEN  = 1;                   // turn on AD again
}

unsigned int readAdc (void)
{
      unsigned int val;

      ADBUSY = 1;
      while (ADBUSY);         //wait until AD conversion complete
      val =  (ADC0H << 8) | ADC0L;
      return val;
}

// le'see, 20Mh clock give 20 cycles per usec
void usDelay (unsigned int val)
{
      unsigned int tprog;
```

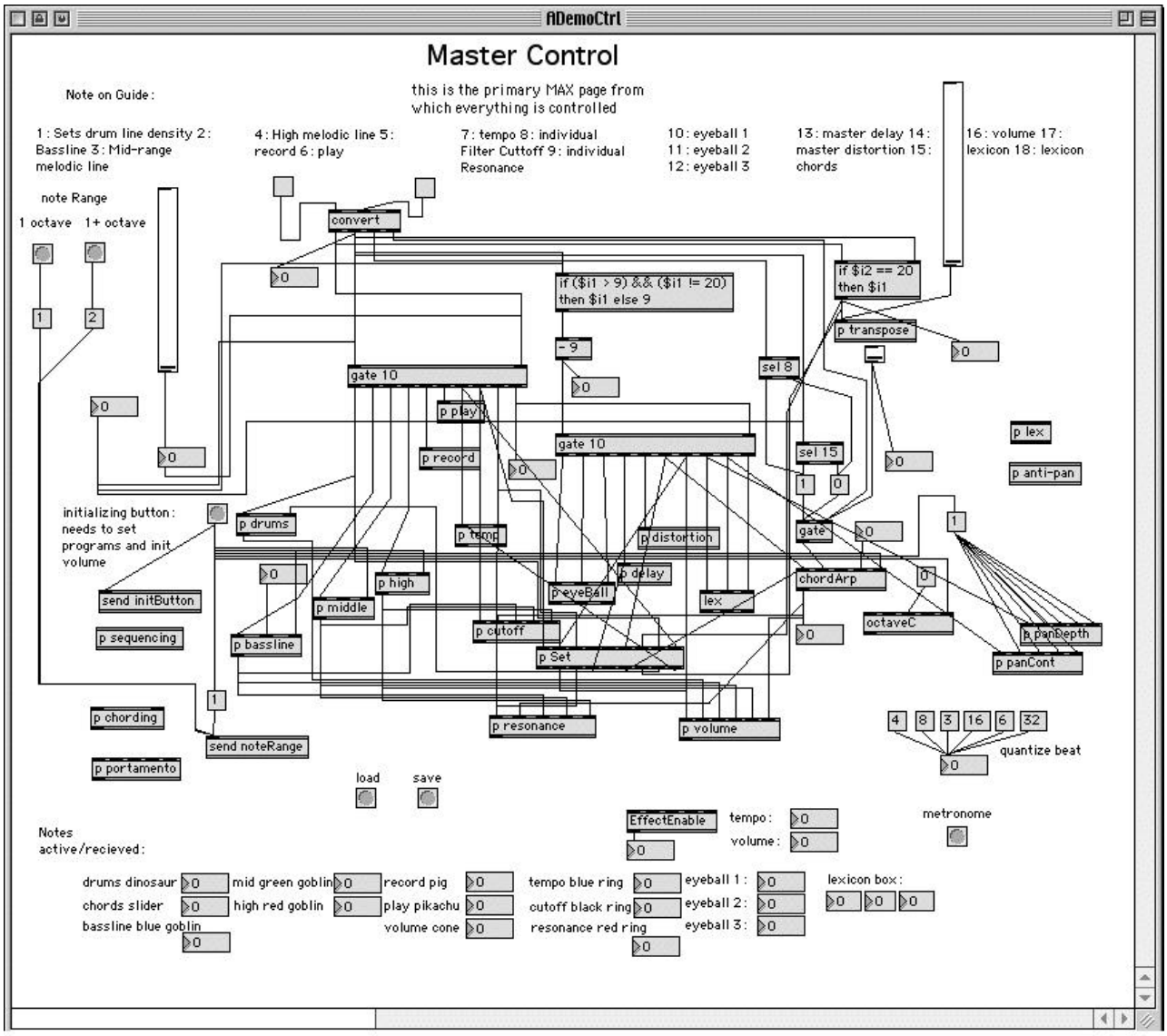
```
      tprog = (65536 - (val - 1));
      TMOD  |= 0x01;  // 16 bit counter
      CKCON |= 0x04;    //use sysclk
      TL0 = (tprog&0x00FF);
      TH0 = (tprog >> 8);
      TR0 = 1;
      while (!TF0);
      TR0 = 0;
      TF0 = 0;
}

void addbuf(char val)
{
  if (bufend==bufstart-1 || (bufend==79 && bufstart==0)) return;
  buffer[bufend++]=val;
  if (bufend>79) bufend=0;
}

char getbuf(void)
{
   char val;
   if (bufend==bufstart) return 255;
   val=buffer[bufstart++];
   if (bufstart>79) bufstart=0;
   return val;
}

void loop (void) interrupt 14
{
      data int tmp;
      data unsigned int pca; // bstmp, betmp;

    TMR3CN = 0x06;            // reset interrupt flag (set to 0x86 for maximum
speed)

//    if (counter==255) {overflows++; counter=0;}
//    else counter++;

    tmp=readAdc();
      pca = PCA0L;
      pca += (PCA0H<<8);

      if (even) {
        basePo++;
        even = false;
        }
    else
        even = true;
    if (calib) {
        tmp = tmp - (baseBuf[basePo] + 600);
        if ((pca>ttag4)&&(ttag4!=0xffff))
          ttag4On();
      else if (pca>ttag3)
            ttag3On();
            else if (pca>ttag2)
                ttag2On();
      }
```

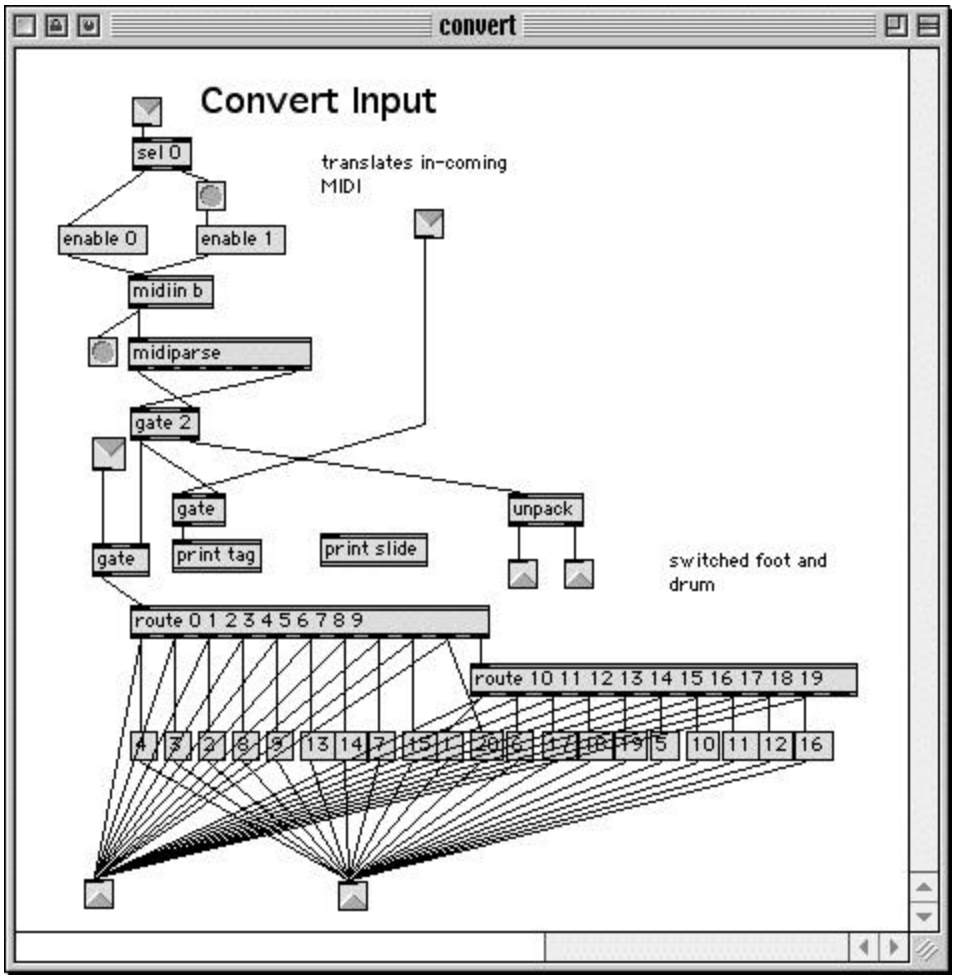
```
      else tmp = tmp - baseBuf[basePo + axisOffset];

        if (tmp>0)
       {
           if (last==0)
            {
               start = pca;
            }
           last=1;
               sum += tmp;
//         sum+=((1/tmp)^(2/3)-.0225)^.5;   // no floating point!!
       }
      else
       {
           if (last==1)
            {
//             addbuf(start_o);
                addbuf(start >> 8);
                        addbuf(start&0x00FF);
                        addbuf(255);      // these 255s are mainly here so
things are
                                          // easier to read in during debug.
                        addbuf(pca >> 8);
                        addbuf(pca&0x00FF);
                        addbuf(255);      // right now, receiving progs. not
expect it.
                  addbuf((int)(sum>>8));
                  addbuf((int)(sum&0x00FF));
                  last=0;
                  sum=0;
                  }
         }

  PIN_D4 = ~PIN_D4;
  if (j >= 4000)
   {        //led blinks every 4000 samples
//      LED = ~LED;
        j = 0;
      }
  j++;
}
```

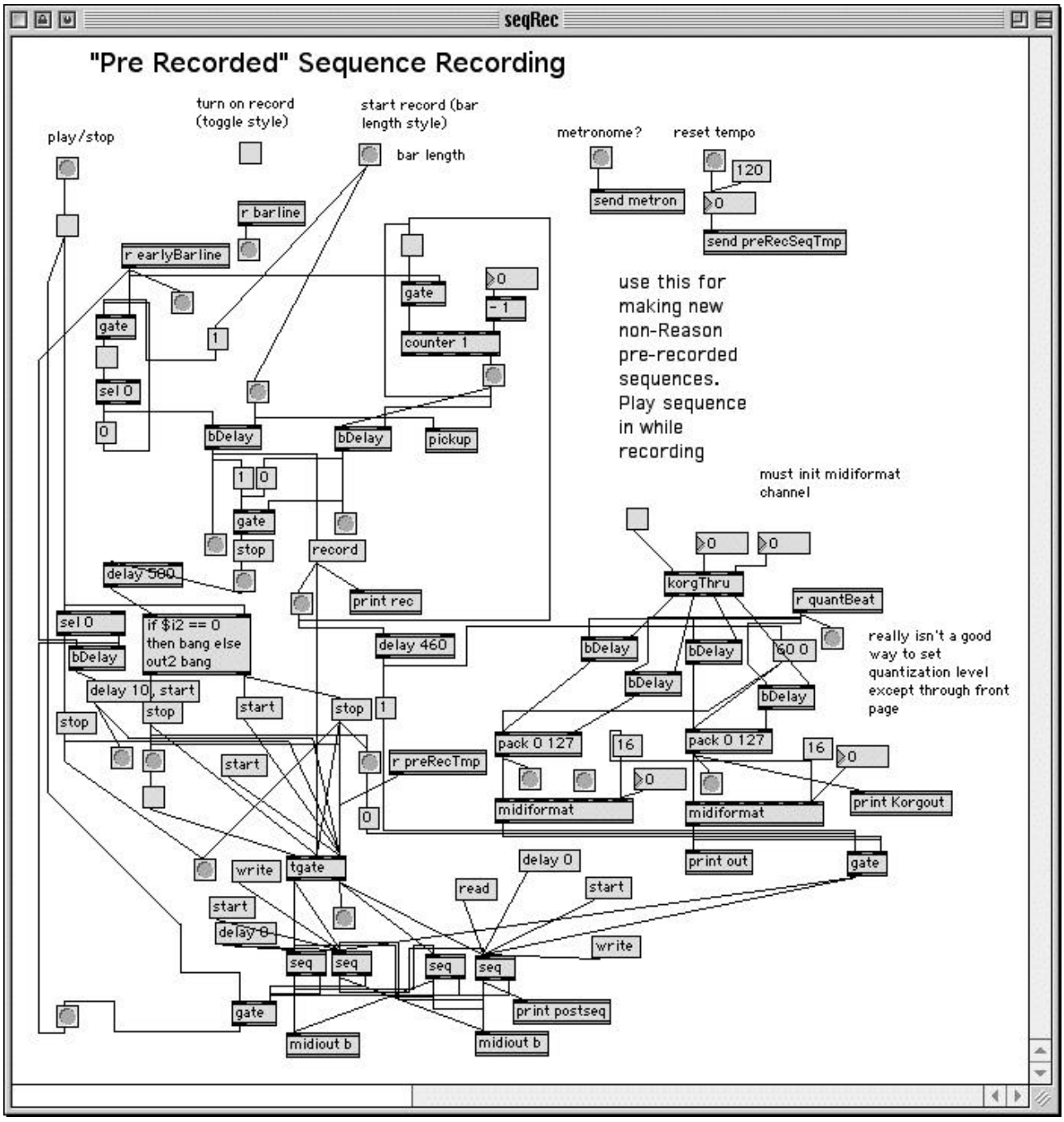
Appendix C: MAX Code

This appendix includes major portions of the *MAX* application code

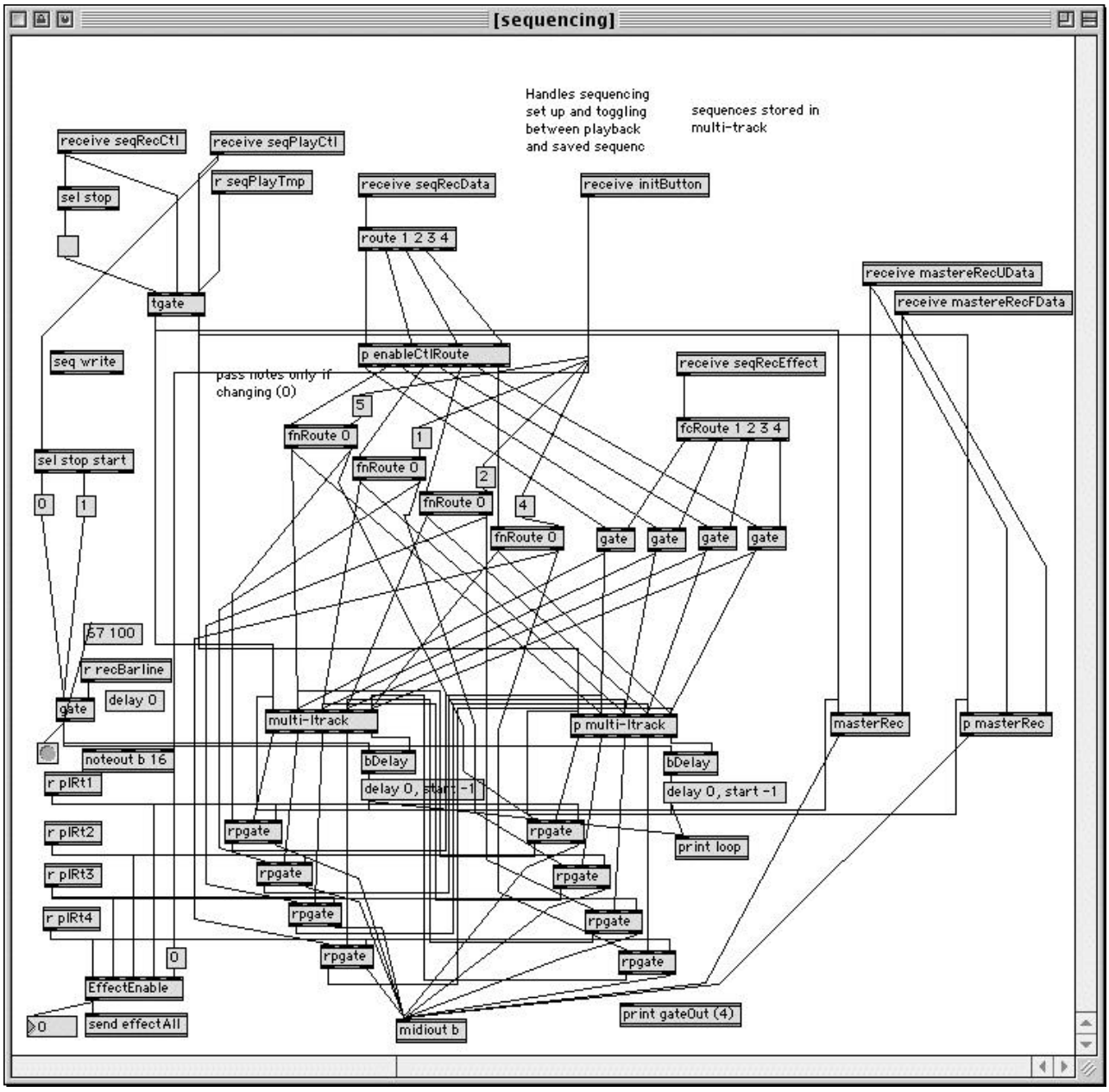
C.A General Patches

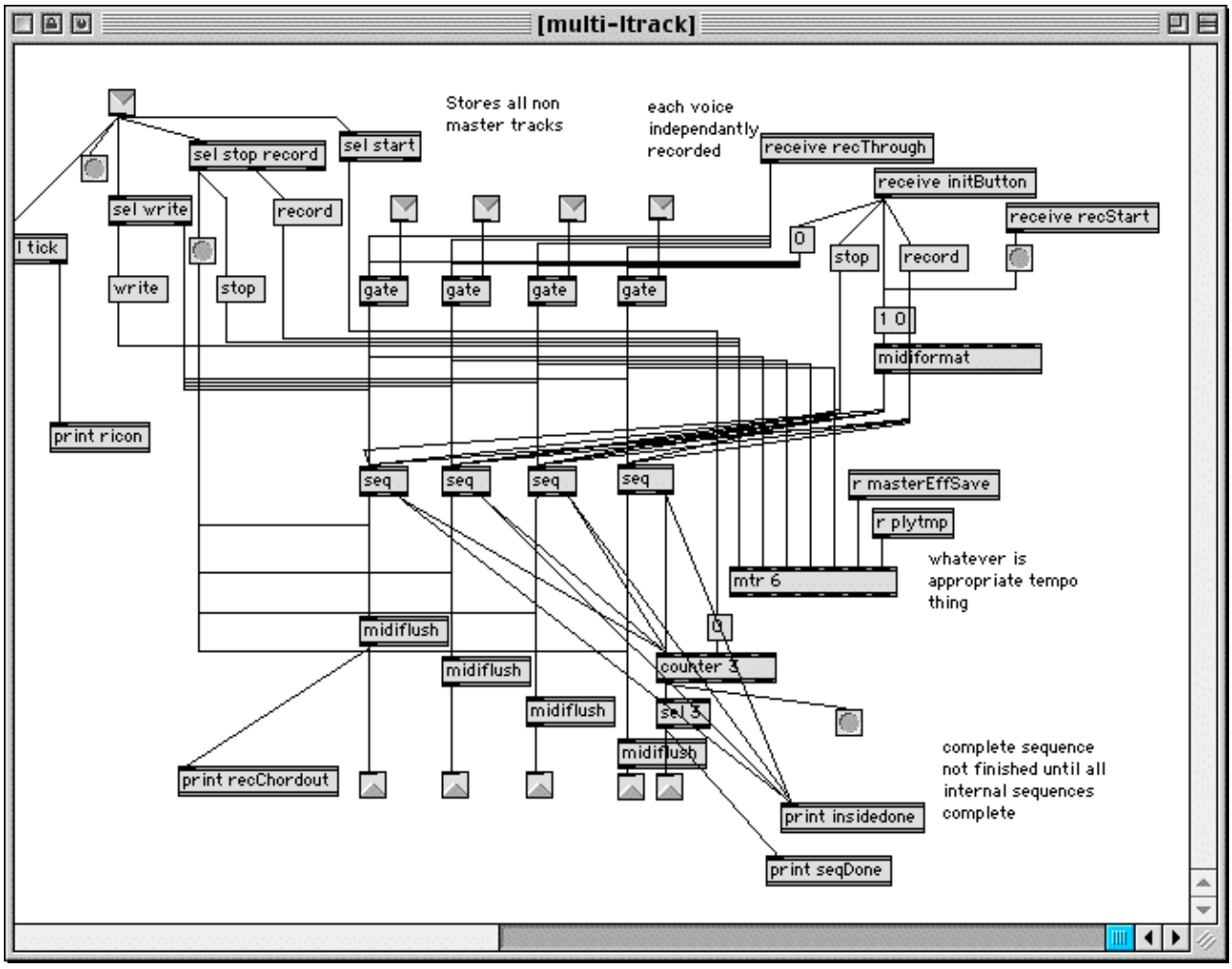


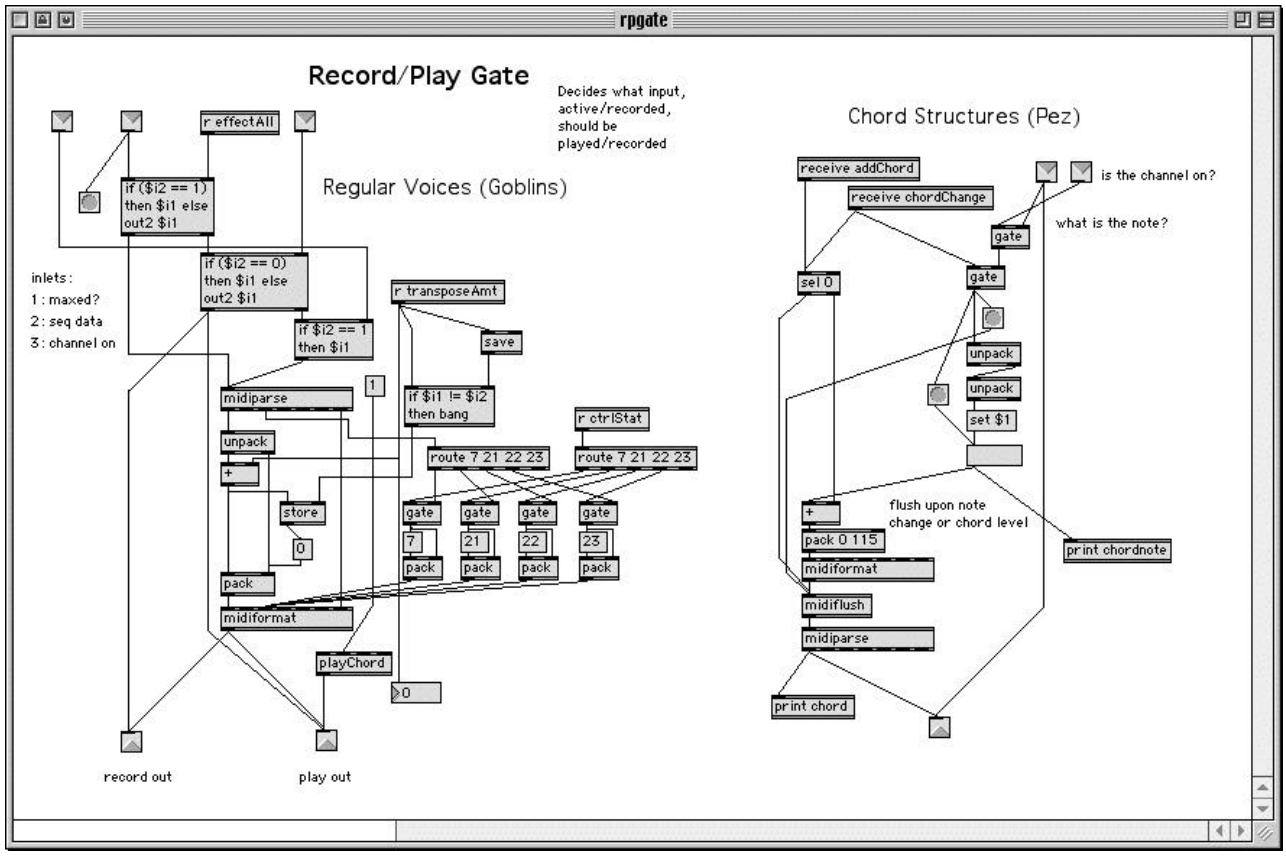

convert
Convert Input
sel 0
translates in-coming
MIDI
enable 0
enable 1
midiin b
midiparse
gate 2
gate
unpack
gate
print tag
print slide
switched foot and
drum
route 0 1 2 3 4 5 6 7 8 9
route 10 11 12 13 14 15 16 17 18 19
4 3 2 6 9 13 14 7 15 1 20 8 17 18 19 5 10 11 12 16

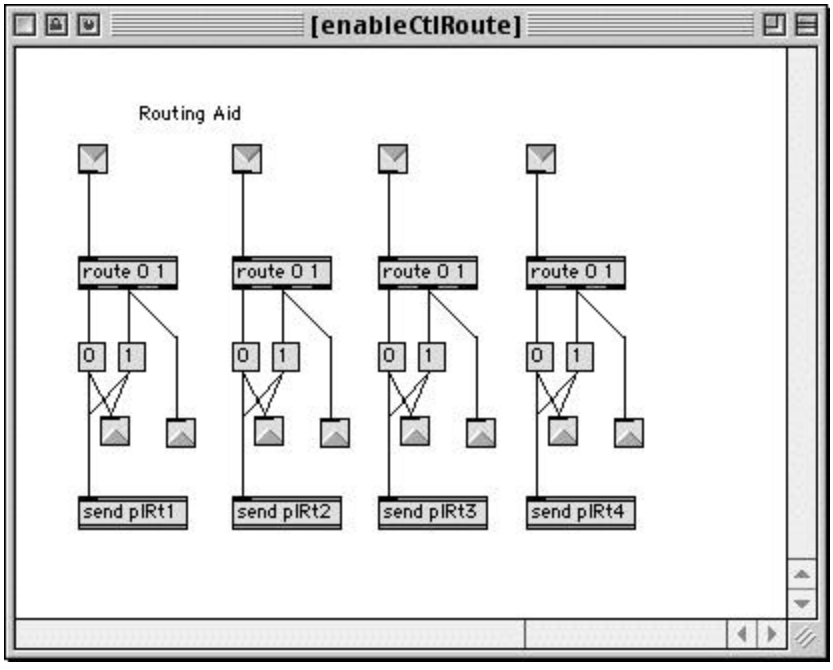

seqRec
"Pre Recorded" Sequence Recording
play/stop
turn on record (toggle style)
start record (bar length style)
bar length
metronome?
reset tempo
120
send metron
send preRecSeqTmp
r barline
r earlyBarline
gate
sel 0
0
1
bDelay
pickup
counter 1
- 1
use this for making new non-Reason pre-recorded sequences. Play sequence in while recording
must init midiformat channel
stop
record
print rec
delay 580
if $i2 == 0 then bang else out2 bang
delay 460
delay 10
start
korgThru
r quantBeat
really isn't a good way to set quantization level except through front page
pack 0 127
16
midiformat
print Korgout
print out
r preRecTmp
write
tgate
read
delay 0
delay 8
seq
print postseq
midiout b

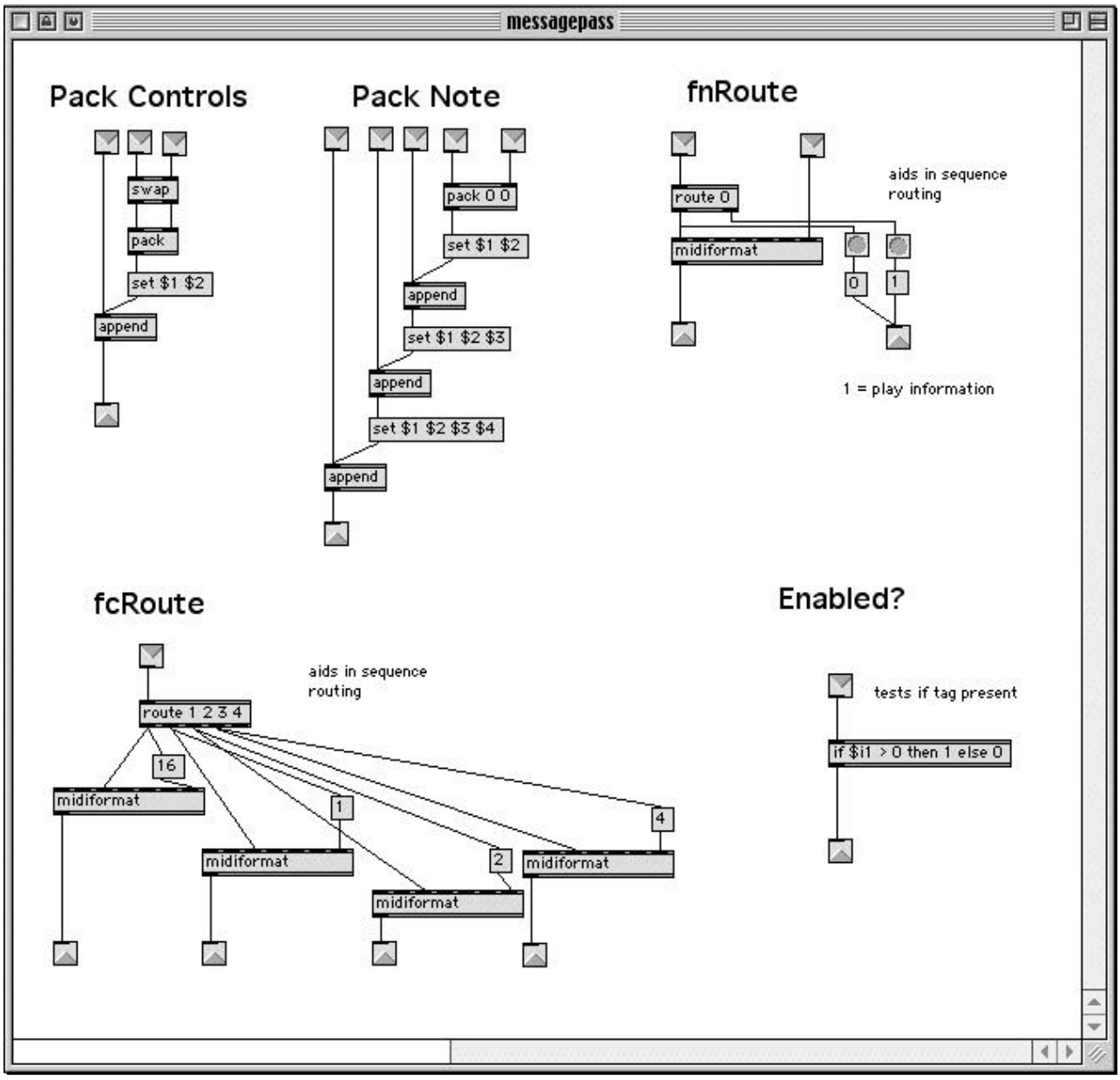
C.B Sequencing Patches

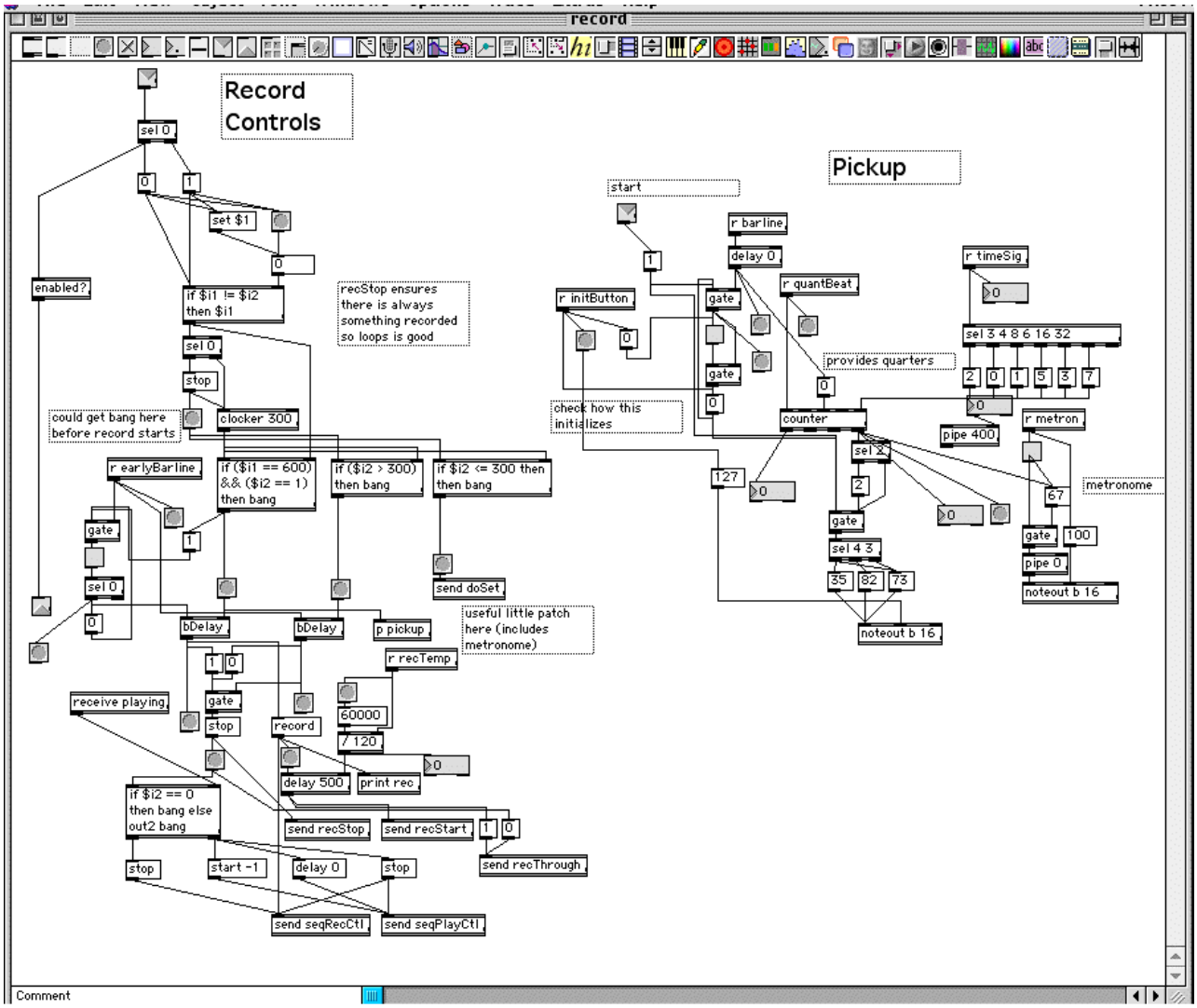


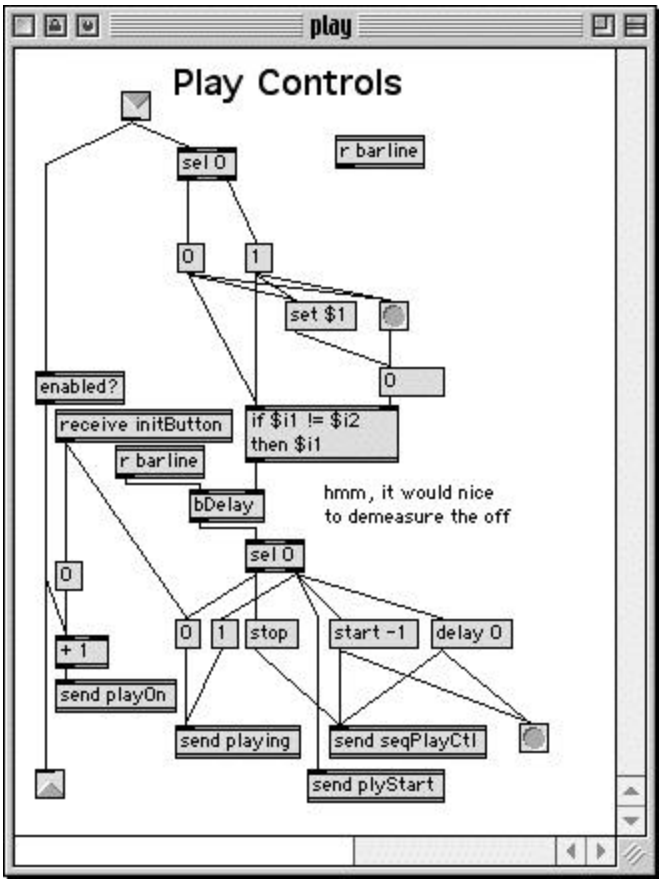

[multi-ltrack]
Stores all non master tracks
each voice independantly recorded
sel stop record
sel start
receive recThrough
receive initButton
receive recStart
sel write
record
l tick
write
stop
gate
gate
gate
gate
0
stop
record
1 0
midiformat
print ricon
seq
seq
seq
seq
r masterEffSave
r plytmp
whatever is appropriate tempo thing
mtr 6
midiflush
0
counter 3
midiflush
midiflush
sel 3
midiflush
print recChordout
complete sequence not finished until all internal sequences complete
print insidedone
print seqDone

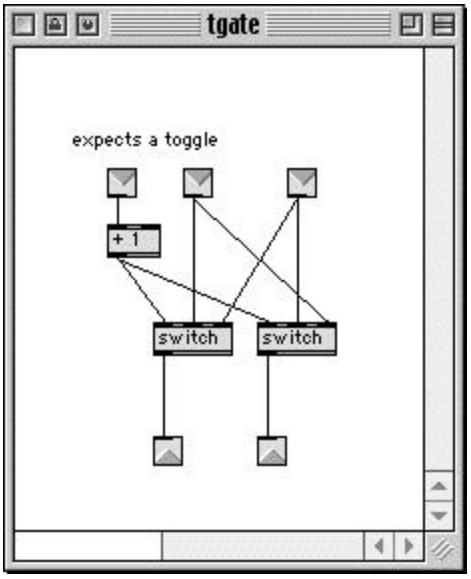

rpgate
Record/Play Gate
Decides what input, active/recorded, should be played/recorded
Regular Voices (Goblins)
Chord Structures (Pez)
inlets:
1 : maxed?
2 : seq data
3 : channel on
r effectAll
if ($i2 == 1) then $i1 else out2 $i1
if ($i2 == 0) then $i1 else out2 $i1
if $i2 == 1 then $i1
r transposeAmt
save
midiparse
unpack
+
store
0
pack
midiformat
playChord
>0
if $i1 != $i2 then bang
r ctrlStat
route 7 21 22 23
gate
7
21
22
23
record out
play out
receive addChord
receive chordChange
is the channel on?
what is the note?
sel 0
unpack
set $1
flush upon note change or chord level
pack 0 115
midiflush
print chordnote
print chord

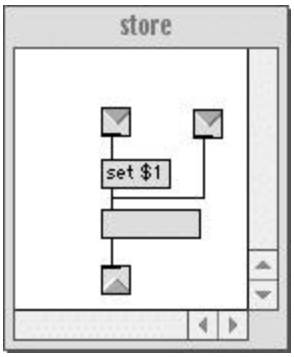

[enableCtlRoute]
Routing Aid
route 0 1
0
1
send plRt1
send plRt2
send plRt3
send plRt4

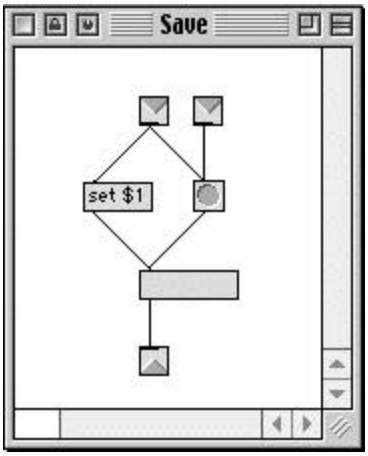

messagepass
Pack Controls
swap
pack
set $1 $2
append
Pack Note
pack 0 0
set $1 $2
append
set $1 $2 $3
append
set $1 $2 $3 $4
append
fnRoute
route 0
midiformat
0
1
aids in sequence routing
1 = play information
fcRoute
aids in sequence routing
route 1 2 3 4
16
midiformat
1
midiformat
2
midiformat
4
midiformat
Enabled?
tests if tag present
if $i1 > 0 then 1 else 0

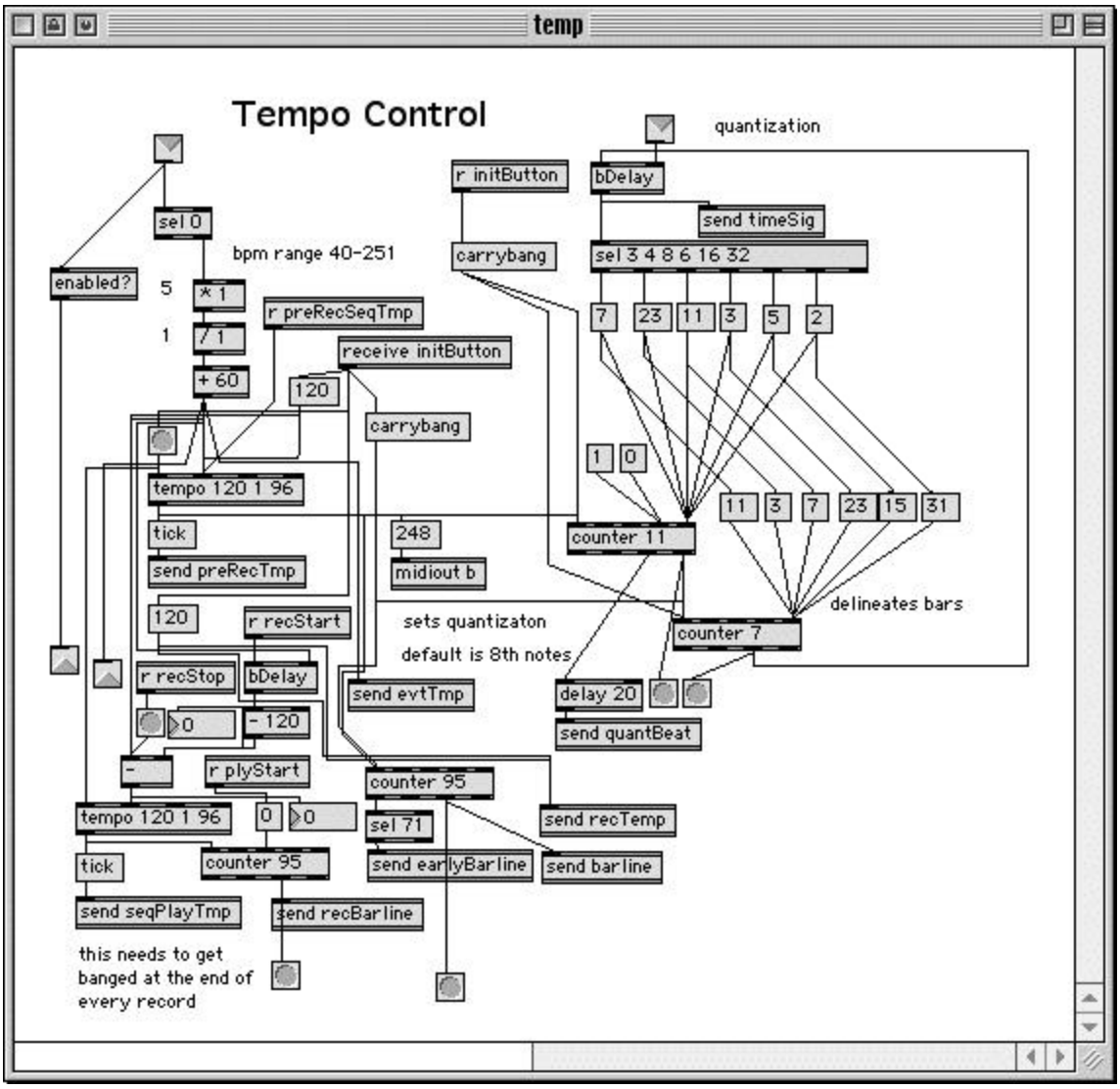

record
Record Controls
sel 0
0
1
set $1
0
enabled?
if $i1 != $i2 then $i1
recStop ensures there is always something recorded so loops is good
sel 0
stop
could get bang here before record starts
clocker 300
r earlyBarline
if ($i1 == 600) && ($i2 == 1) then bang
if ($i2 > 300) then bang
if $i2 <= 300 then then bang
gate
1
sel 0
0
send doSet
bDelay
bDelay
p pickup
useful little patch here (includes metronome)
1
0
r recTemp
receive playing
gate
stop
record
60000
/ 120
>0
delay 500
print rec
if $i2 == 0 then bang else out2 bang
send recStop
send recStart
1
0
send recThrough
stop
start -1
delay 0
stop
send seqRecCtl
send seqPlayCtl
Pickup
start
1
r barline
delay 0
r initButton
0
gate
gate
0
r quantBeat
provides quarters
check how this initializes
counter
127
>0
sel 2
2
gate
sel 4 3
35
82
73
>0
r timeSig
>0
sel 3 4 8 6 16 32
2
0
1
5
3
7
>0
pipe 400
r metron
67
metronome
gate
100
pipe 0
noteout b 16
noteout b 16
Comment


play
Play Controls
r barline
sel 0
0
1
set $1
0
enabled?
receive initButton
if $i1 != $i2
then $i1
r barline
bDelay
hmm, it would nice
to demeasure the off
sel 0
0
0
1
stop
start -1
delay 0
+ 1
send playOn
send playing
send seqPlayCtl
send plyStart

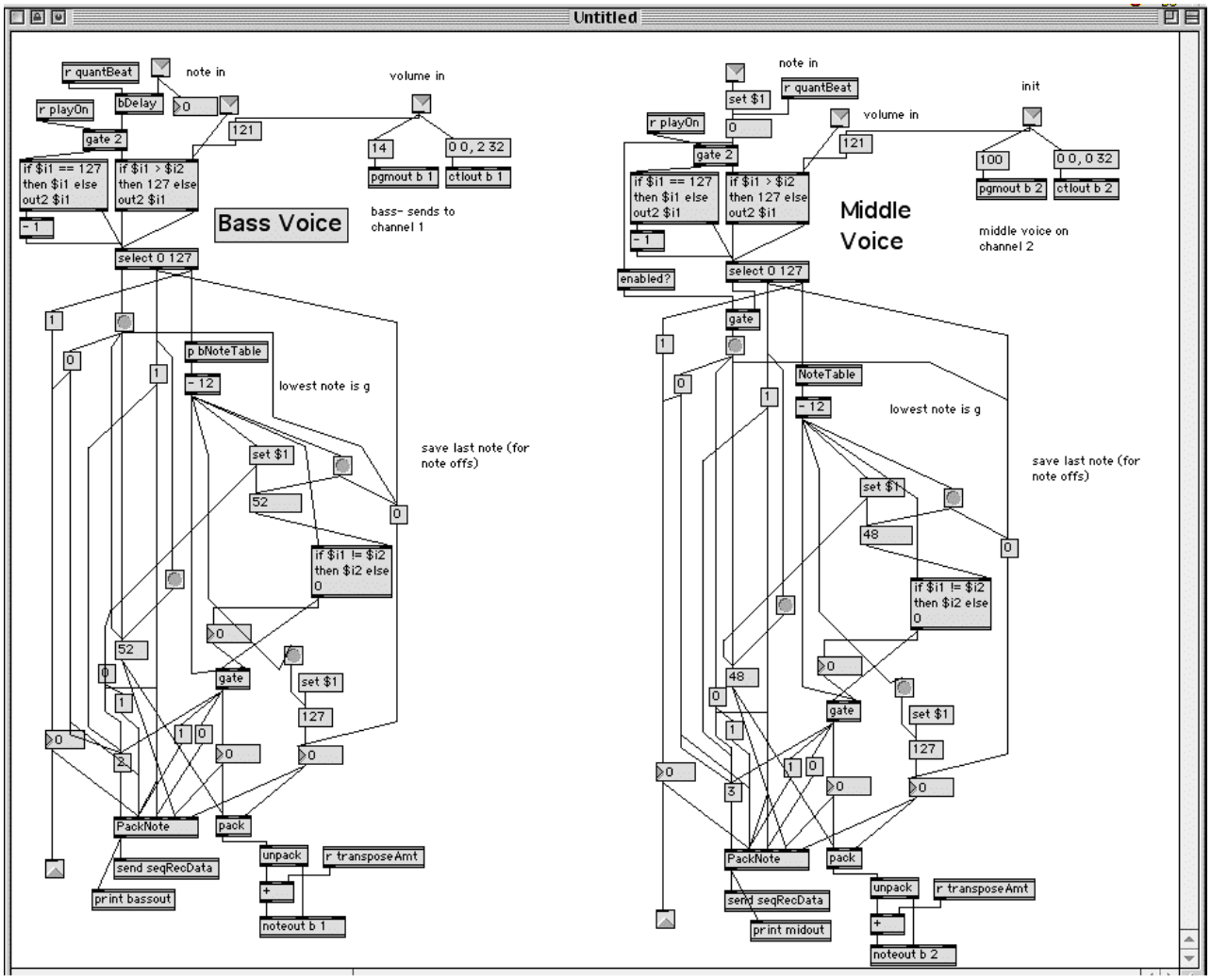

tgate
expects a toggle
+ 1
switch
switch

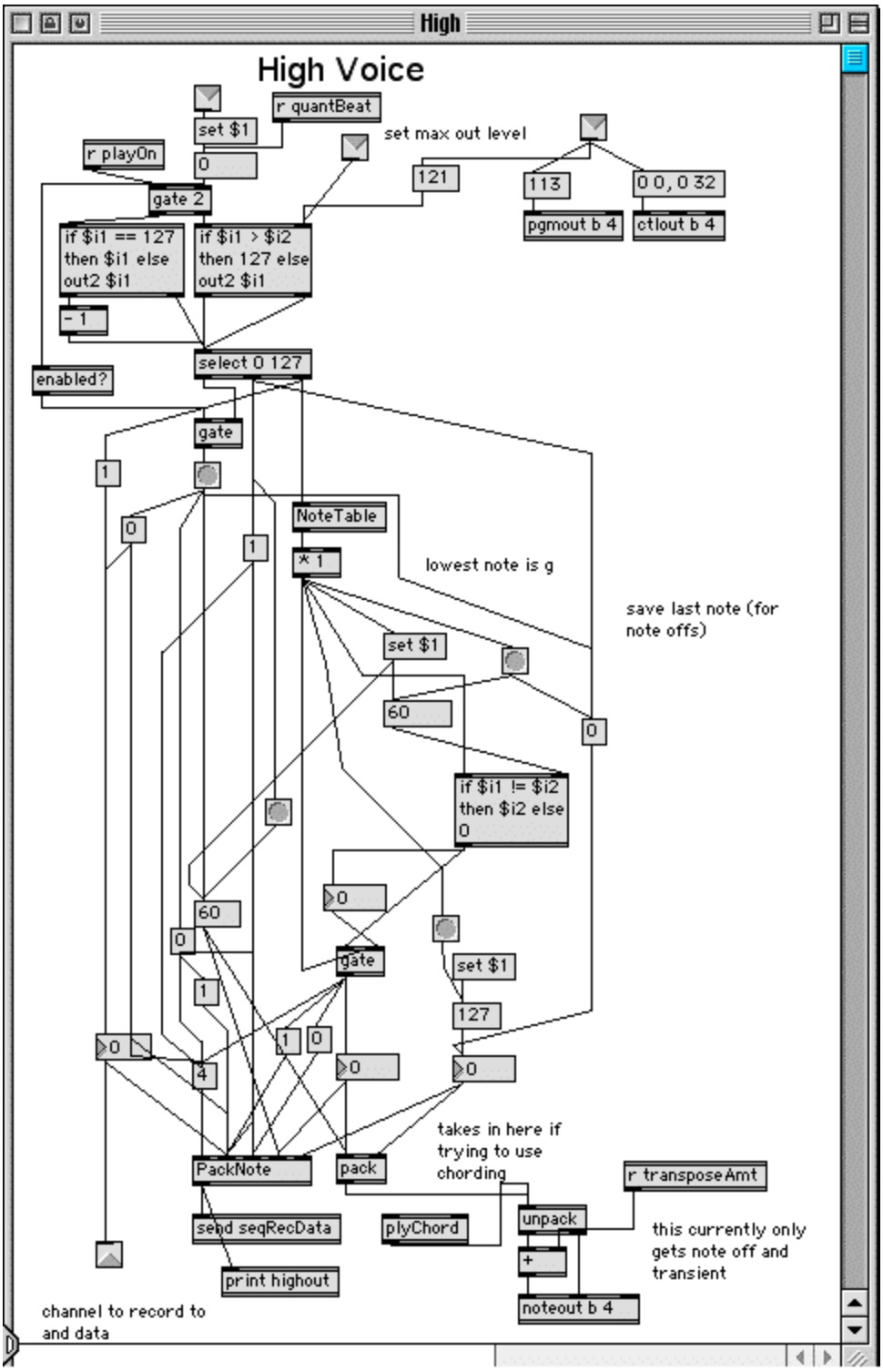

store
set $1

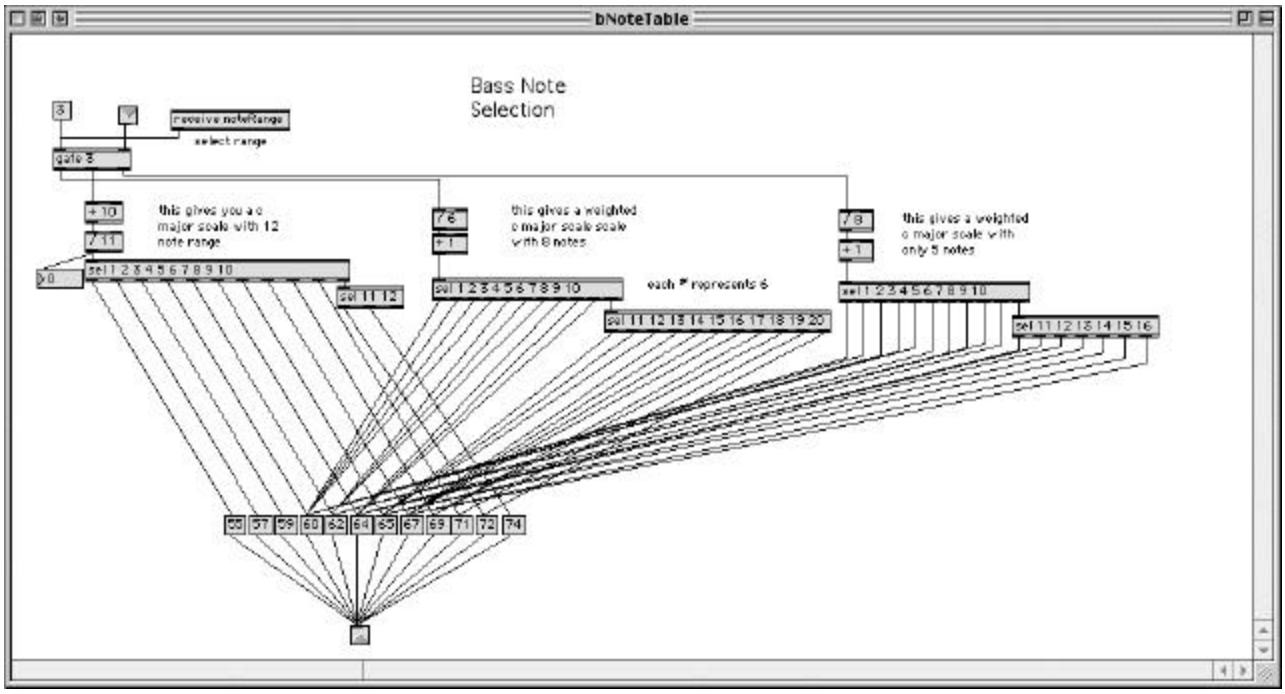

Save
set $1

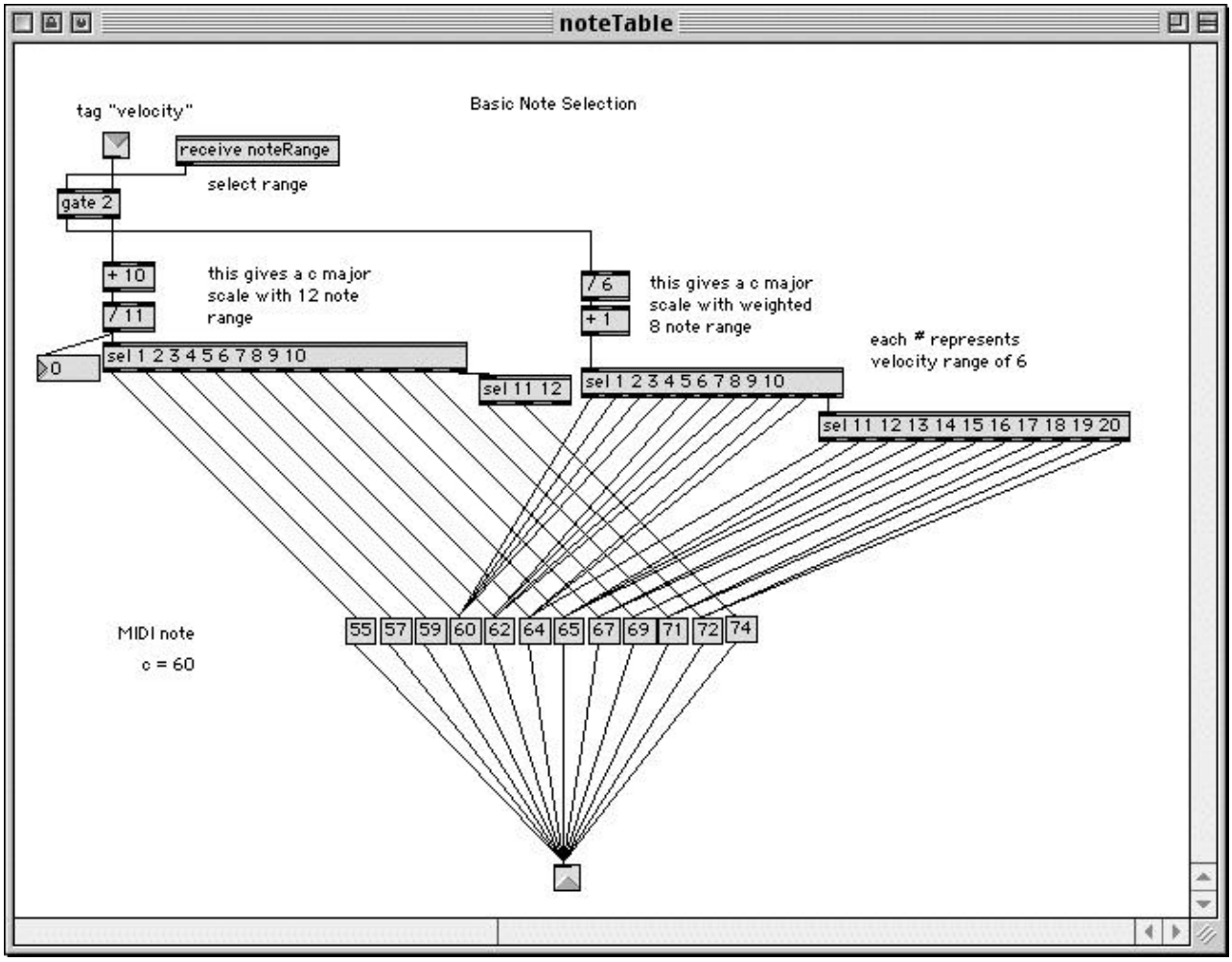

temp
Tempo Control
quantization
r initButton
bDelay
send timeSig
sel 0
bpm range 40-251
carrybang
sel 3 4 8 6 16 32
enabled?
5
* 1
r preRecSeqTmp
7
23
11
3
5
2
1
/ 1
receive initButton
+ 60
120
carrybang
1
0
tempo 120 1 96
11
3
7
23
15
31
tick
248
counter 11
send preRecTmp
midiout b
delineates bars
120
r recStart
sets quantizaton
default is 8th notes
counter 7
r recStop
bDelay
send evtTmp
delay 20
0
- 120
send quantBeat
-
r plyStart
counter 95
tempo 120 1 96
0
0
sel 71
send recTemp
tick
counter 95
send earlyBarline
send barline
send seqPlayTmp
send recBarline
this needs to get
banged at the end of
every record

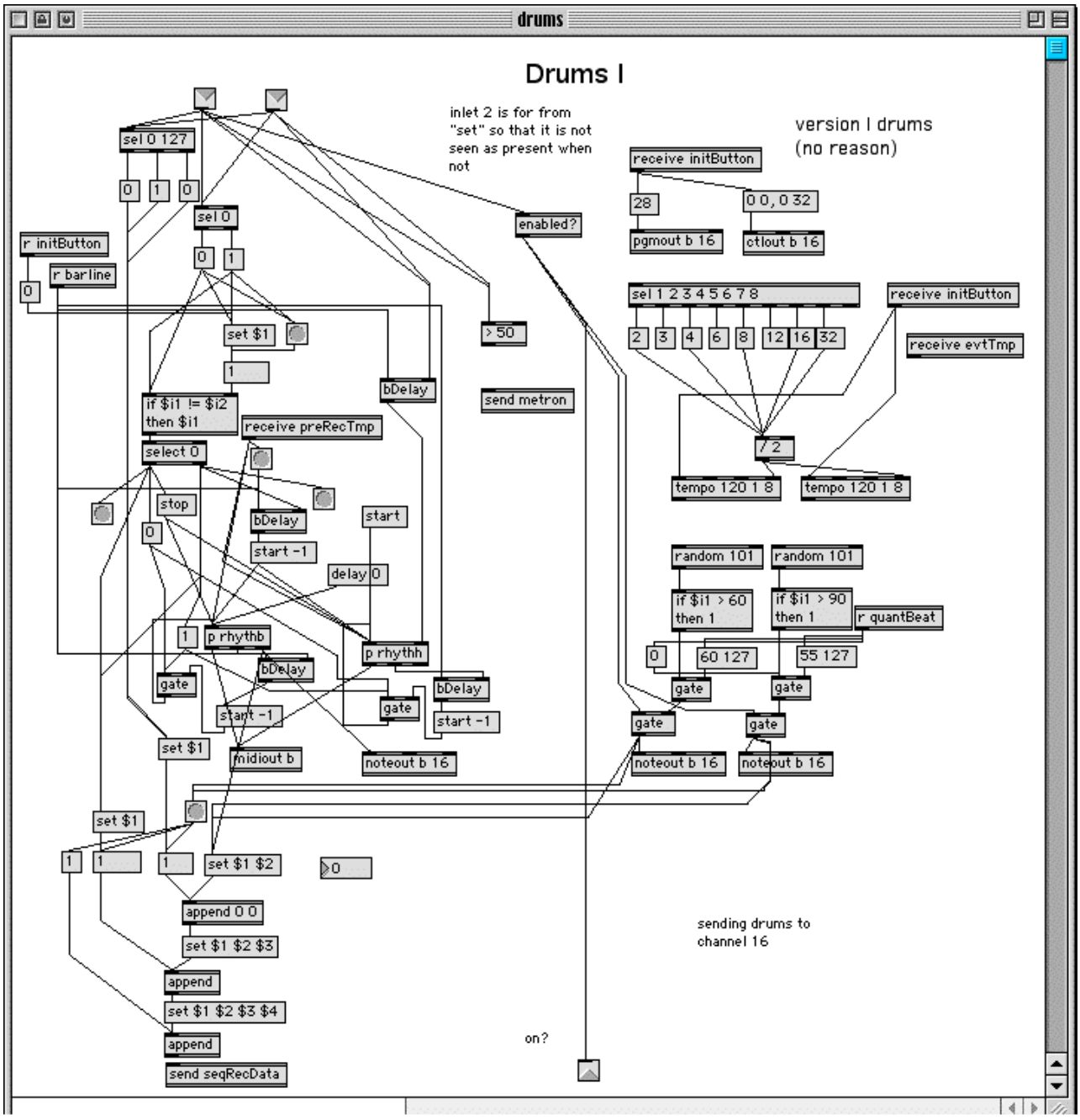
C.C Voice Patches

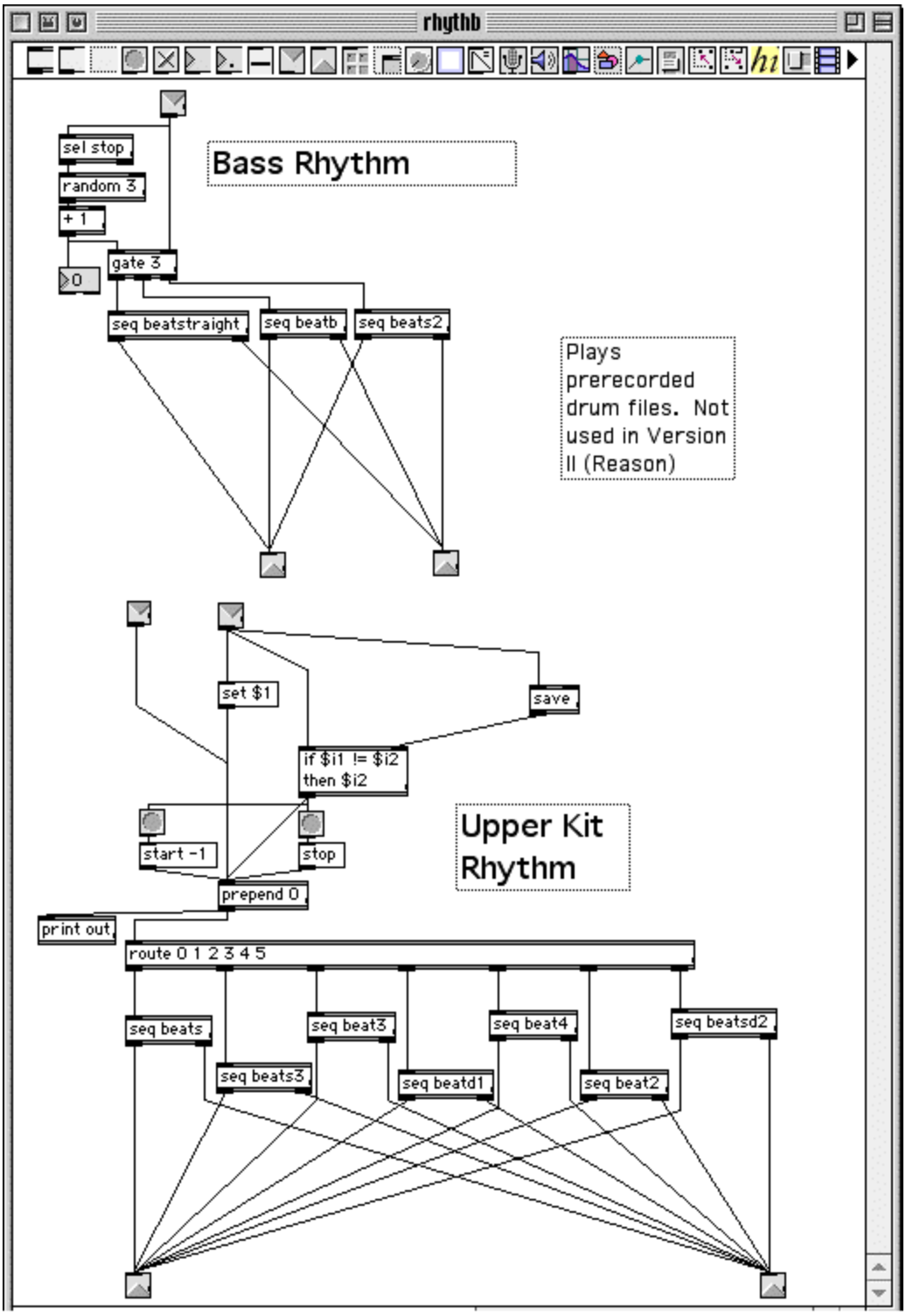


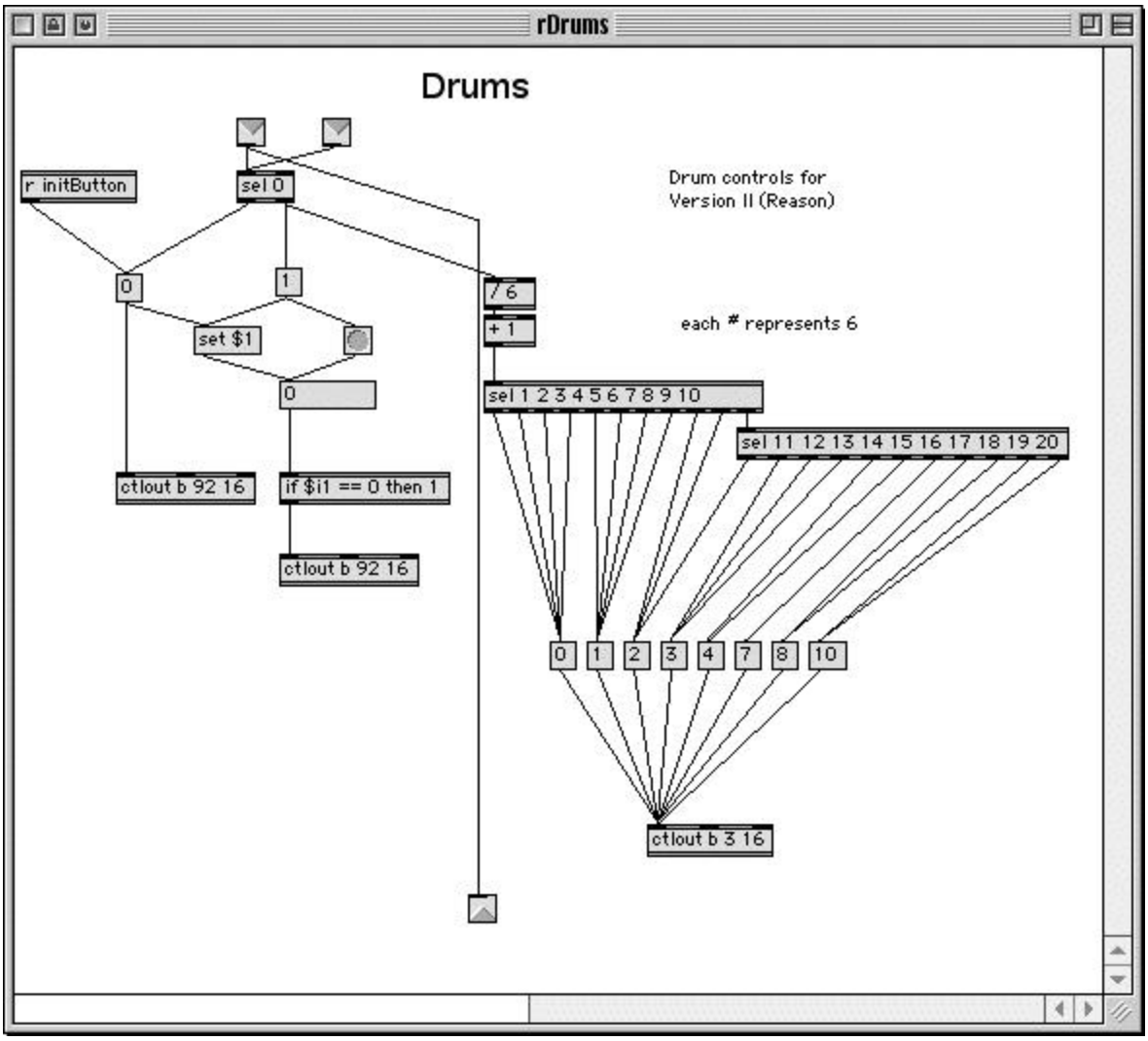

High
High Voice
r quantBeat
set $1
0
r playOn
gate 2
set max out level
121
113
0 0, 0 32
pgmout b 4
ctlout b 4
if $i1 == 127 then $i1 else out2 $i1
if $i1 > $i2 then 127 else out2 $i1
- 1
select 0 127
enabled?
gate
1
0
1
NoteTable
* 1
lowest note is g
save last note (for note offs)
set $1
60
0
if $i1 != $i2 then $i2 else 0
0
60
0
gate
set $1
127
1
0
1
0
4
0
0
takes in here if trying to use chording
PackNote
pack
r transposeAmt
send seqRecData
plyChord
unpack
this currently only gets note off and transient
print highout
+
noteout b 4
channel to record to and data

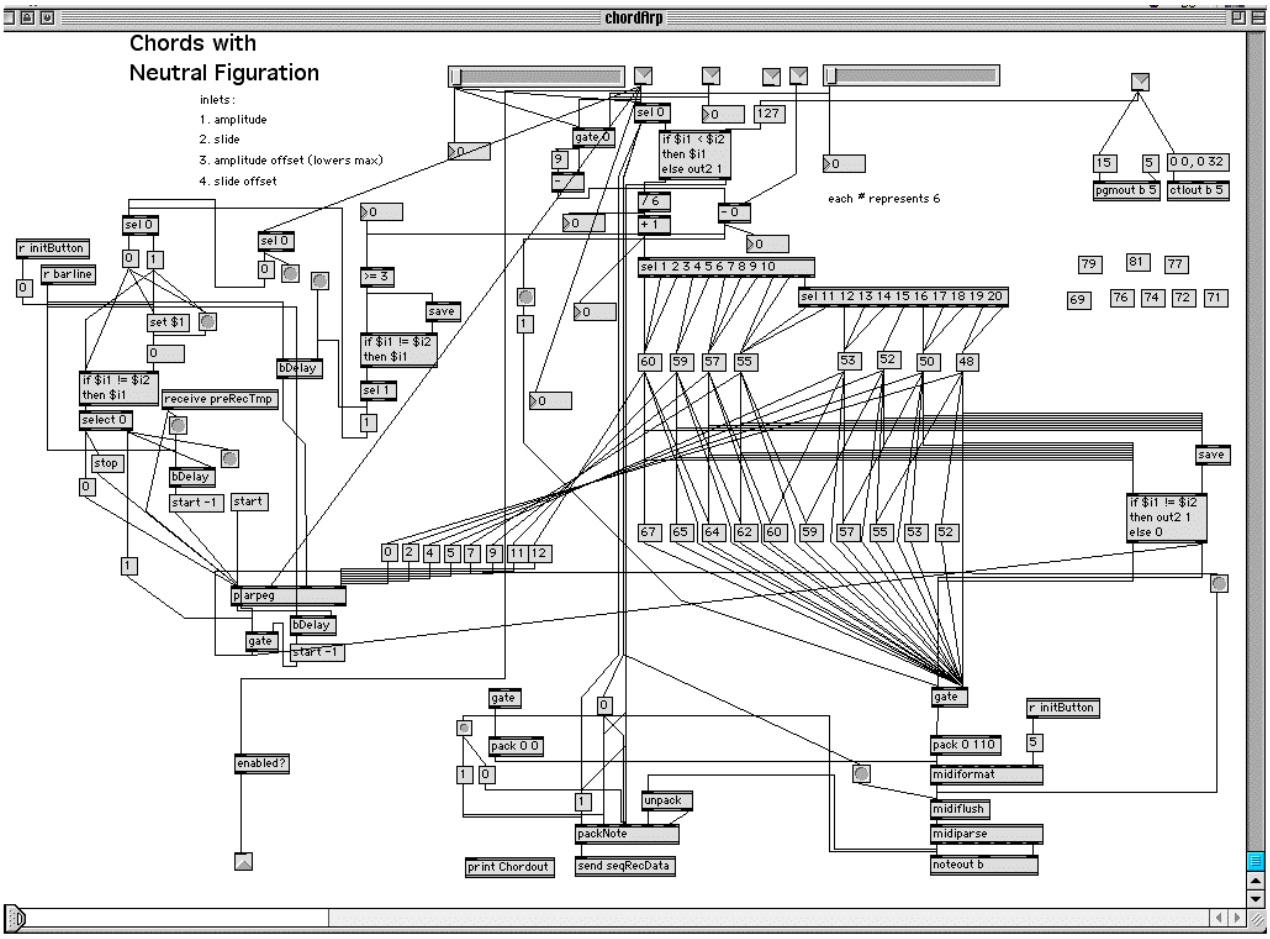

bNoteTable
Bass Note Selection
receive noteRange
select range
gate 8
+ 10
/ 11
this gives you a c major scale with 12 note range
/ 6
+ 1
this gives a weighted c major scale scale with 8 notes
/ 8
+ 1
this gives a weighted c major scale with only 5 notes
each # represents 6
sel 1 2 3 4 5 6 7 8 9 10
sel 11 12
sel 11 12 13 14 15 16 17 18 19 20
55 57 59 60 62 64 65 67 69 71 72 74

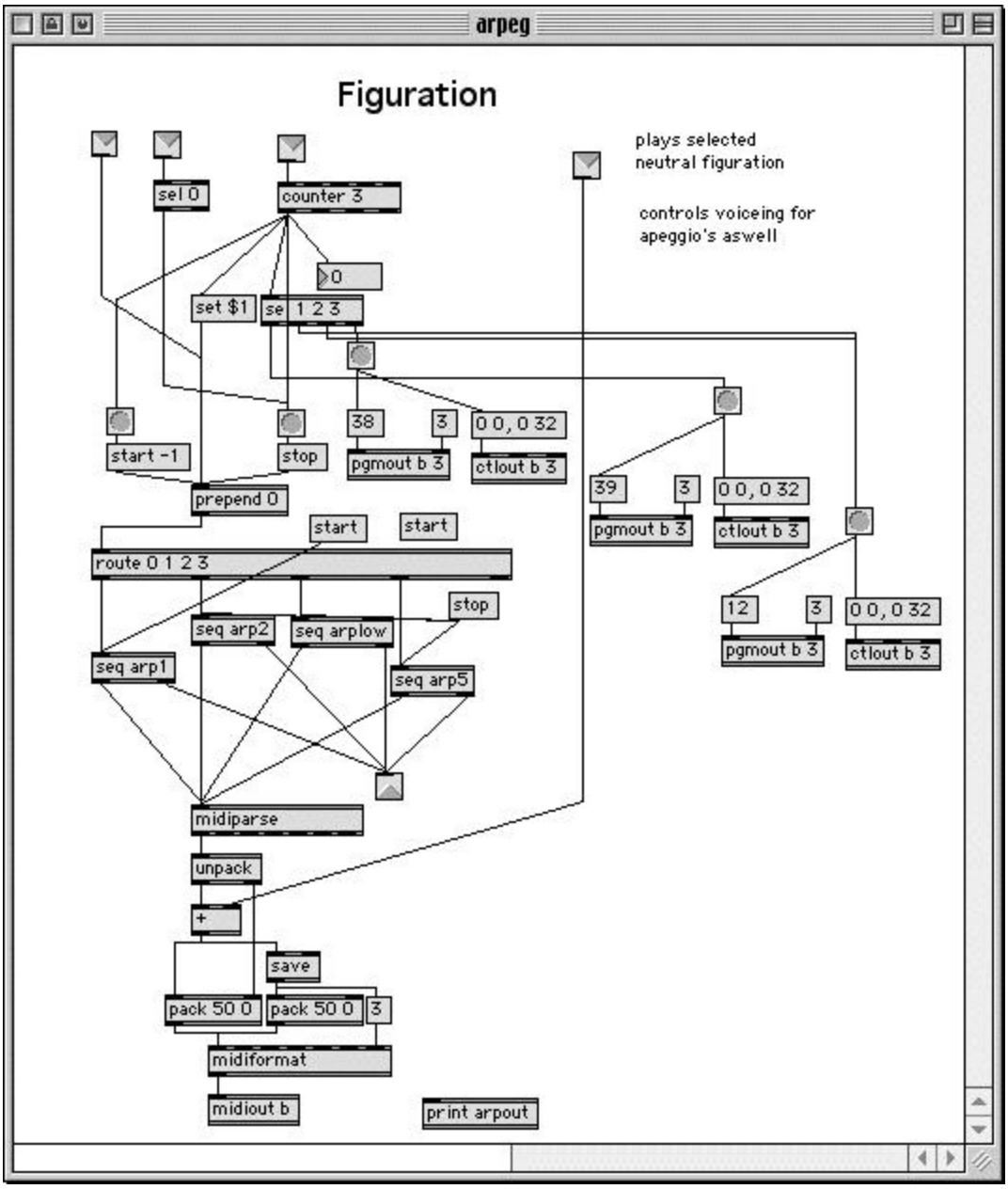

noteTable
tag "velocity"
Basic Note Selection
receive noteRange
select range
gate 2
+ 10
/ 11
this gives a c major scale with 12 note range
/ 6
+ 1
this gives a c major scale with weighted 8 note range
each # represents velocity range of 6
sel 1 2 3 4 5 6 7 8 9 10
sel 11 12
sel 11 12 13 14 15 16 17 18 19 20
MIDI note
c = 60
55 57 59 60 62 64 65 67 69 71 72 74

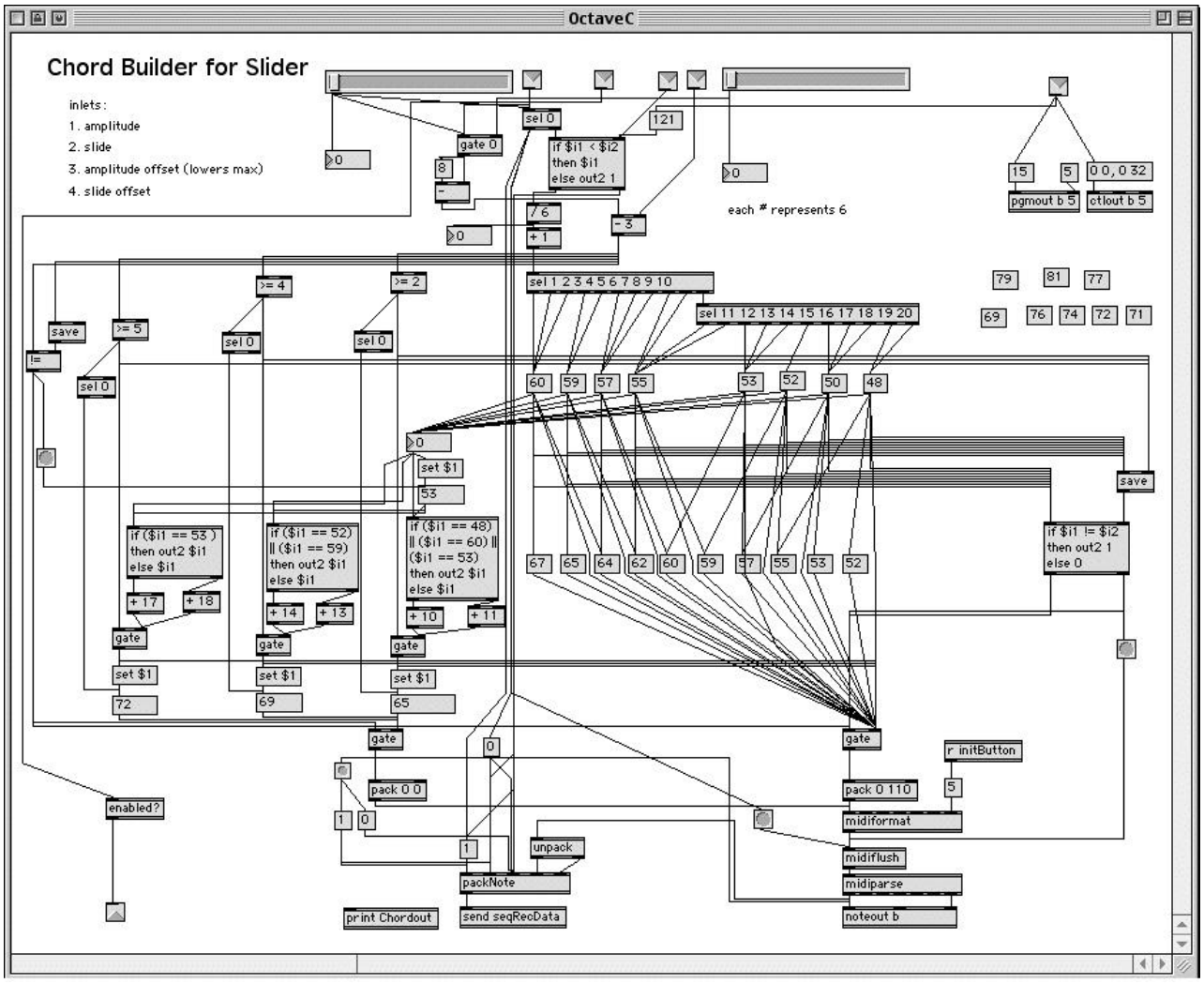

drums
Drums I
inlet 2 is for from "set" so that it is not seen as present when not
version I drums (no reason)
sel 0 127
0
1
0
sel 0
r initButton
r barline
0
0
1
set $1
1
bDelay
if $i1 != $i2 then $i1
receive preRecTmp
select 0
stop
0
bDelay
start
start -1
delay 0
1
p rhythb
bDelay
gate
start -1
gate
bDelay
start -1
p rhythh
set $1
midiout b
noteout b 16
set $1
1
1
1
set $1 $2
>0
append 0 0
set $1 $2 $3
append
set $1 $2 $3 $4
append
send seqRecData
enabled?
> 50
send metron
on?
receive initButton
28
pgmout b 16
0 0, 0 32
ctlout b 16
sel 1 2 3 4 5 6 7 8
2
3
4
6
8
12
16
32
receive initButton
receive evtTmp
/ 2
tempo 120 1 8
tempo 120 1 8
random 101
random 101
if $i1 > 60 then 1
if $i1 > 90 then 1
r quantBeat
0
60 127
55 127
gate
gate
gate
gate
noteout b 16
noteout b 16
sending drums to channel 16

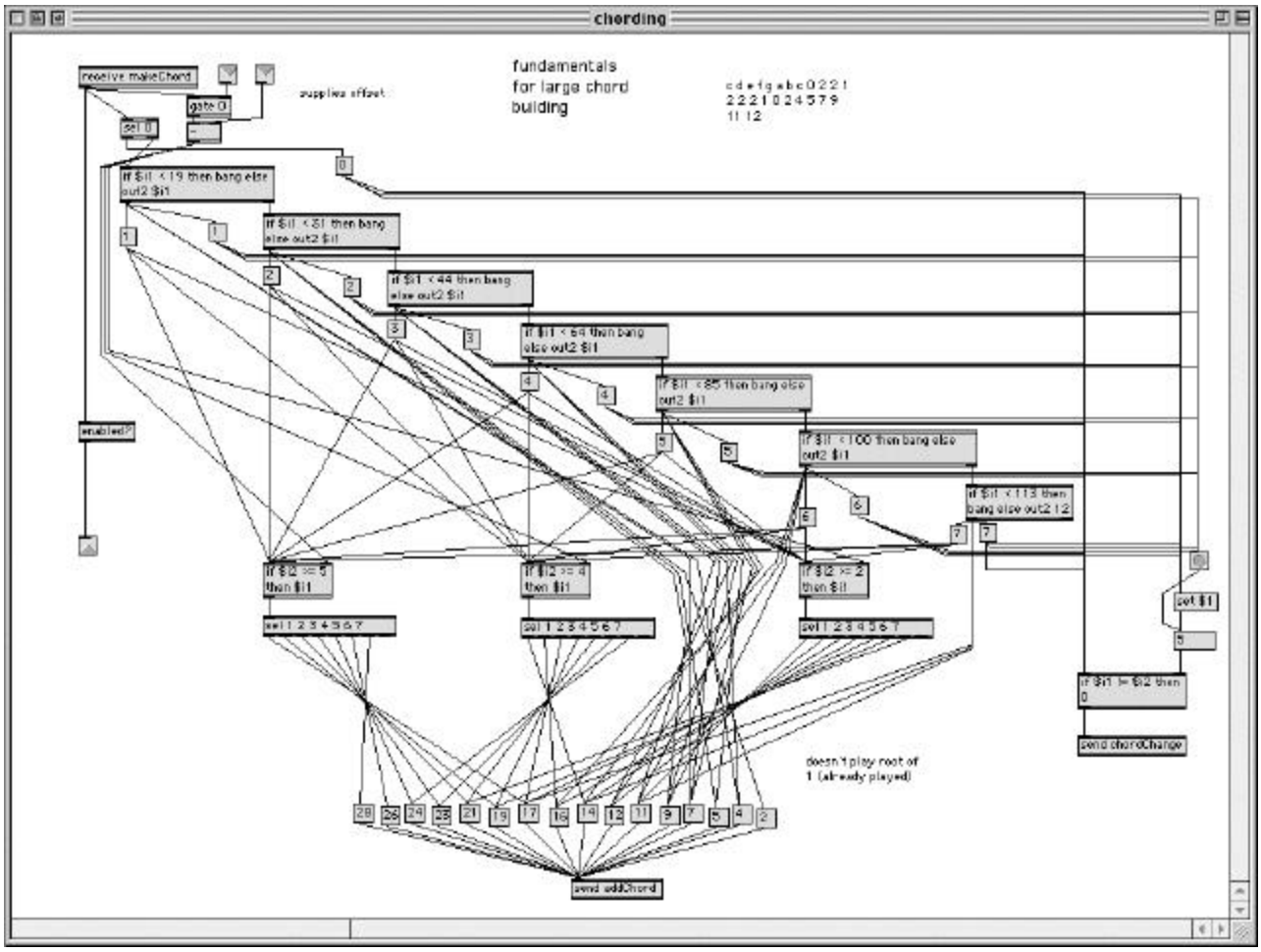

rhythb
sel stop
random 3
+ 1
gate 3
0
Bass Rhythm
seq beatstraight
seq beatb
seq beats2
Plays prerecorded drum files. Not used in Version II (Reason)
set $1
save
if $i1 != $i2 then $i2
start -1
stop
prepend 0
print out
Upper Kit Rhythm
route 0 1 2 3 4 5
seq beats
seq beat3
seq beat4
seq beatsd2
seq beats3
seq beatd1
seq beat2


rDrums
Drums
r initButton
sel 0
Drum controls for
Version II (Reason)
0
1
/ 6
+ 1
each # represents 6
set $1
0
sel 1 2 3 4 5 6 7 8 9 10
sel 11 12 13 14 15 16 17 18 19 20
ctlout b 92 16
if $i1 == 0 then 1
ctlout b 92 16
0
1
2
3
4
7
8
10
ctlout b 3 16

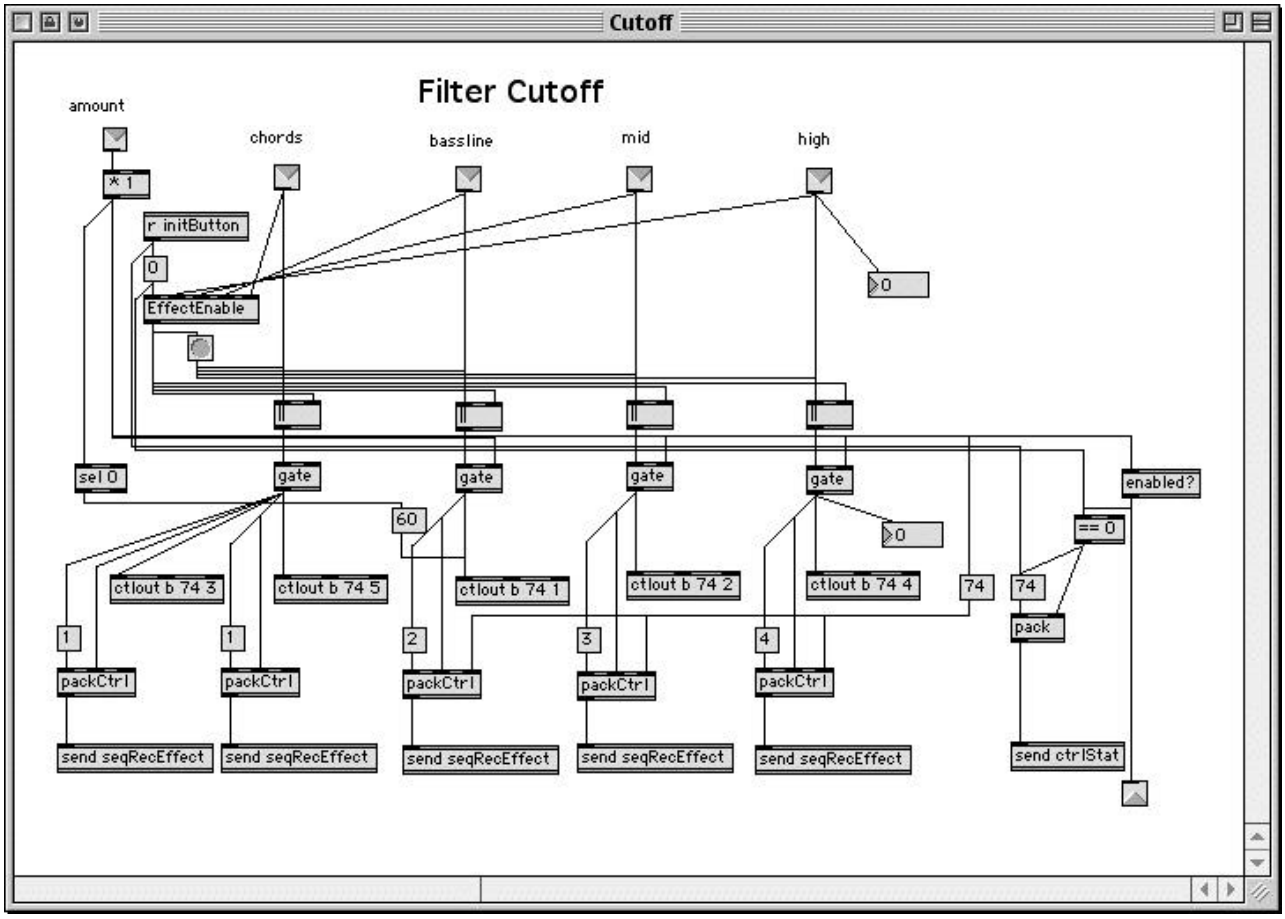

chordfirp
Chords with
Neutral Figuration
inlets :
1. amplitude
2. slide
3. amplitude offset (lowers max)
4. slide offset
each # represents 6
r initButton
r barline
sel 0
set $1
bDelay
if $i1 != $i2
then $i1
receive preRecTmp
select 0
stop
start -1
start
save
sel 1
sel 1 2 3 4 5 6 7 8 9 10
sel 11 12 13 14 15 16 17 18 19 20
60 59 57 55
53 52 50 48
67 65 64 62 60 59 57 55 53 52
0 2 4 5 7 9 11 12
parpeg
gate
enabled?
pack 0 0
unpack
packNote
print Chordout
send seqRecData
if $i1 < $i2
then $i1
else out2 1
if $i1 != $i2
then out2 1
else 0
pack 0 110
midiformat
midiflush
midiparse
noteout b
pgmout b 5
ctlout b 5
15
5
0 0, 0 32
79 81 77
69 76 74 72 71

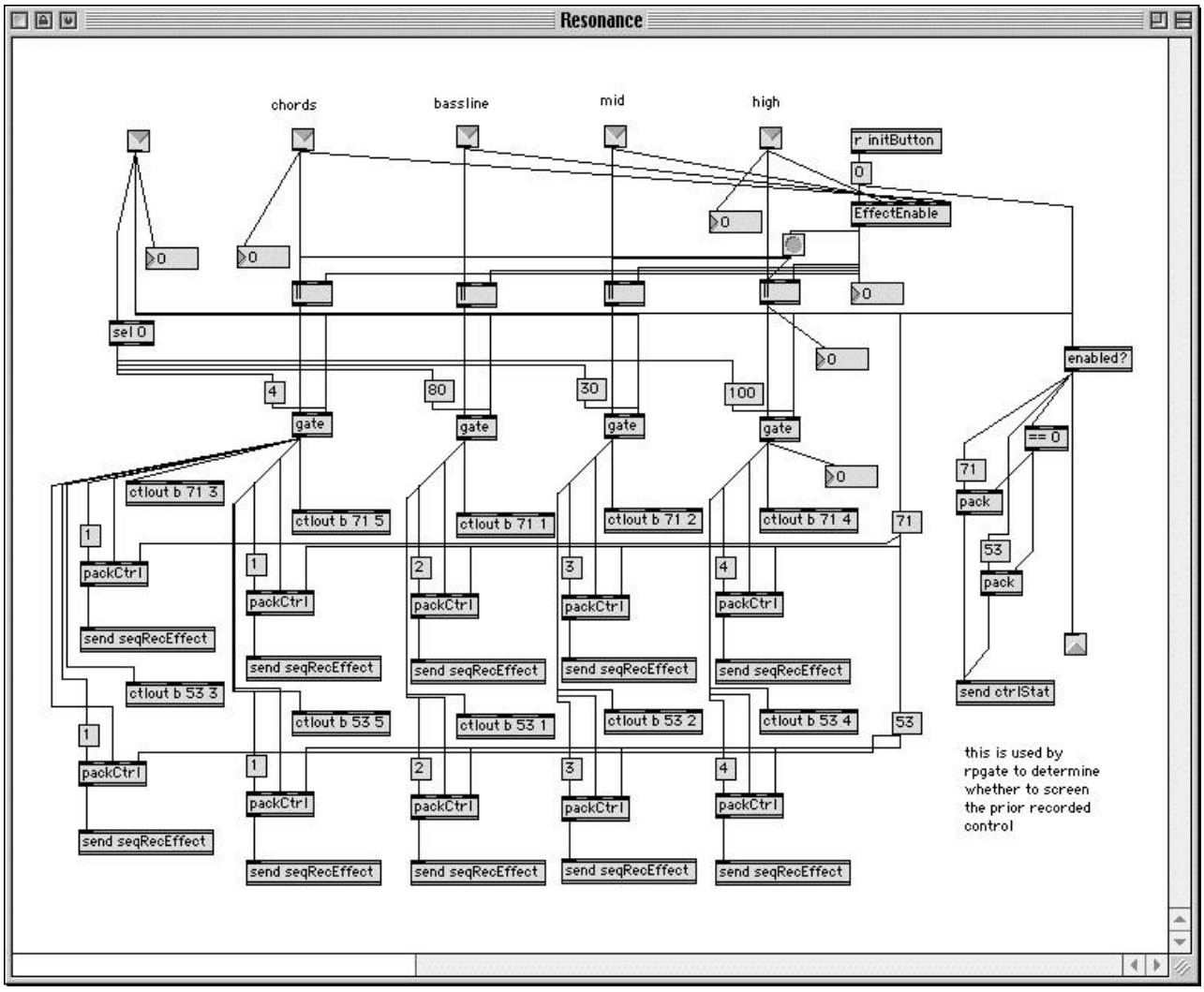

arpeg
Figuration
plays selected
neutral figuration
controls voiceing for
apeggio's aswell
sel 0
counter 3
0
set $1
se 1 2 3
start -1
stop
38
3
0 0, 0 32
pgmout b 3
ctlout b 3
prepend 0
start
start
route 0 1 2 3
stop
seq arp2
seq arplow
seq arp1
seq arp5
39
3
0 0, 0 32
pgmout b 3
ctlout b 3
12
3
0 0, 0 32
pgmout b 3
ctlout b 3
midiparse
unpack
+
save
pack 50 0
pack 50 0
3
midiformat
midiout b
print arpout

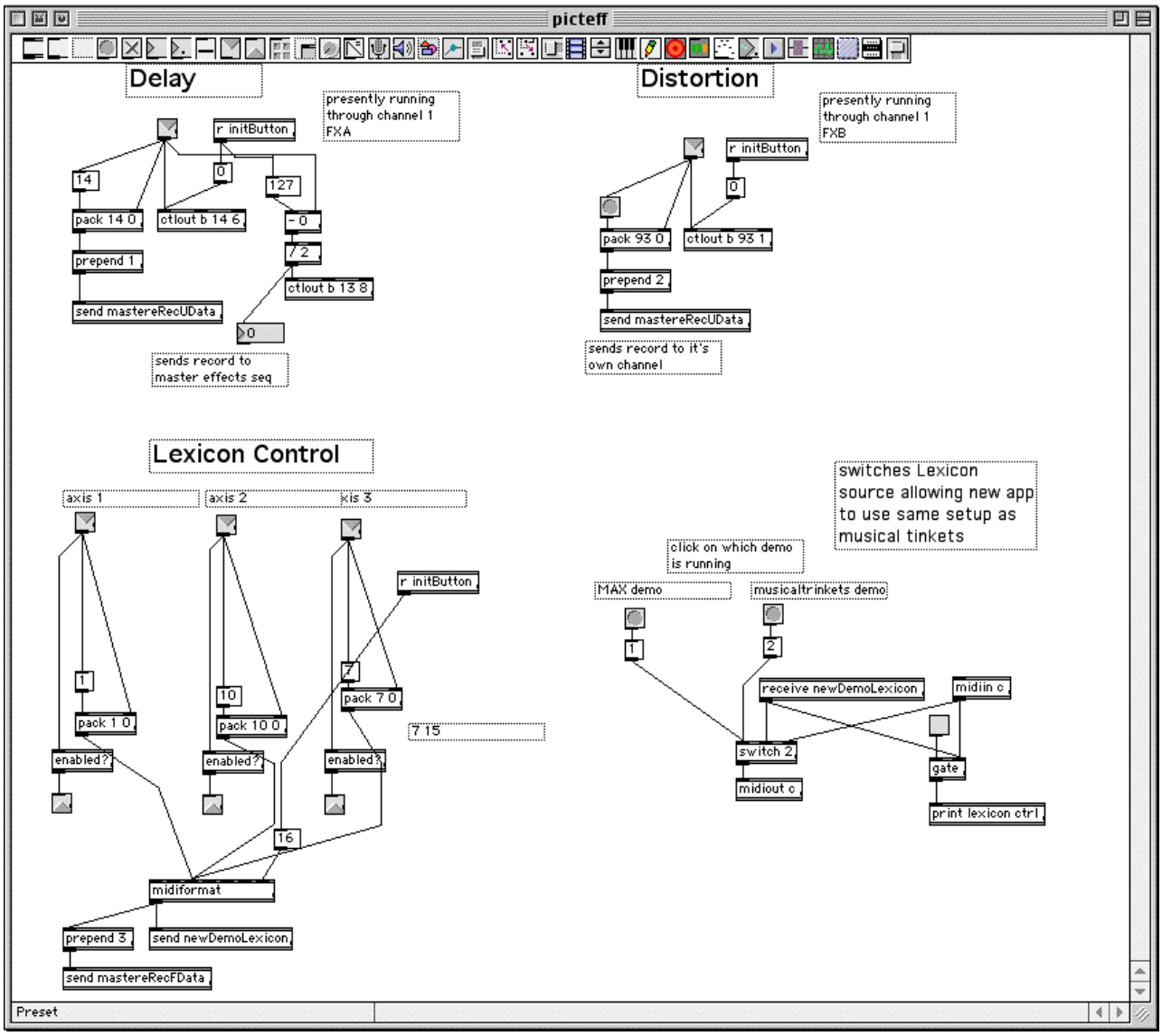

OctaveC
Chord Builder for Slider
inlets:
1. amplitude
2. slide
3. amplitude offset (lowers max)
4. slide offset
each # represents 6
sel 1 2 3 4 5 6 7 8 9 10
sel 11 12 13 14 15 16 17 18 19 20
if ($i1 == 53) then out2 $i1 else $i1
if ($i1 == 52) || ($i1 == 59) then out2 $i1 else $i1
if ($i1 == 48) || ($i1 == 60) || ($i1 == 53) then out2 $i1 else $i1
if $i1 != $i2 then out2 1 else 0
if $i1 < $i2 then $i1 else out2 1
pack 0 0
pack 0 110
packNote
unpack
midiformat
midiflush
midiparse
noteout b
pgmout b 5
ctlout b 5
r initButton
enabled?
print Chordout
send seqRecData
gate
save
set $1

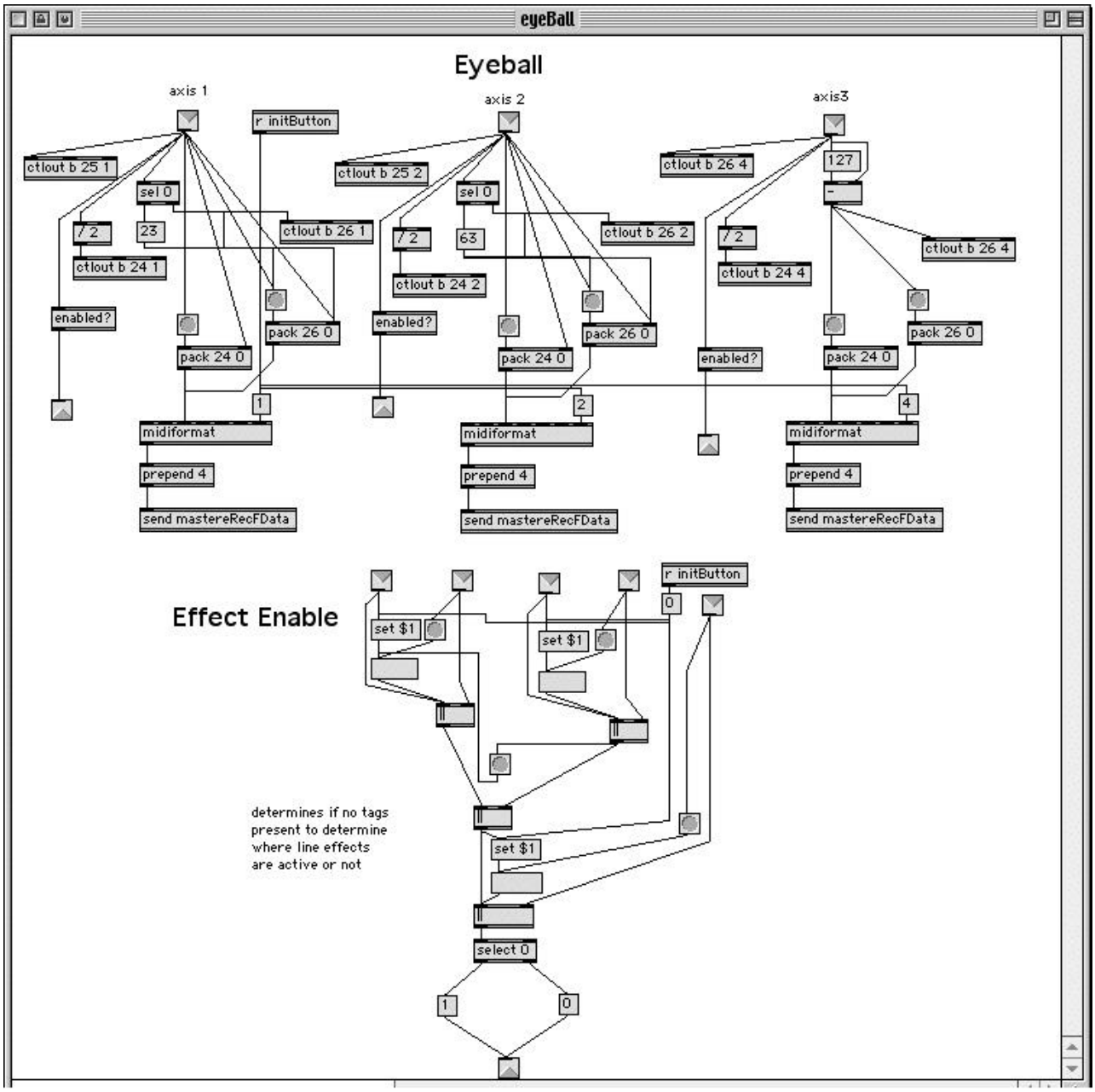

chording
receive makeChord
supplies offset
fundamentals
for large chord
building
c d e f g a b c' 0 2 2 1
2 2 2 1 0 2 4 5 7 9
11 12
gate 0
set 0
if $i1 < 19 then bang else out2 $i1
if $i1 < 51 then bang else out2 $i1
if $i1 < 44 then bang else out2 $i1
if $i1 < 64 then bang else out2 $i1
if $i1 < 85 then bang else out2 $i1
if $i1 < 100 then bang else out2 $i1
if $i1 < 113 then bang else out2 12
enabled?
if $i2 == 0 then $i1
if $i2 == 4 then $i1
if $i2 == 2 then $i1
sel 1 2 3 4 5 6 7
sel 1 2 3 4 5 6 7
sel 1 2 3 4 5 6 7
set $i1
if $i1 != $i2 then 0
send chordChange
doesn't play root of 1 (already played)
28 26 24 23 21 19 17 16 14 12 11 9 7 5 4 2
send addChord

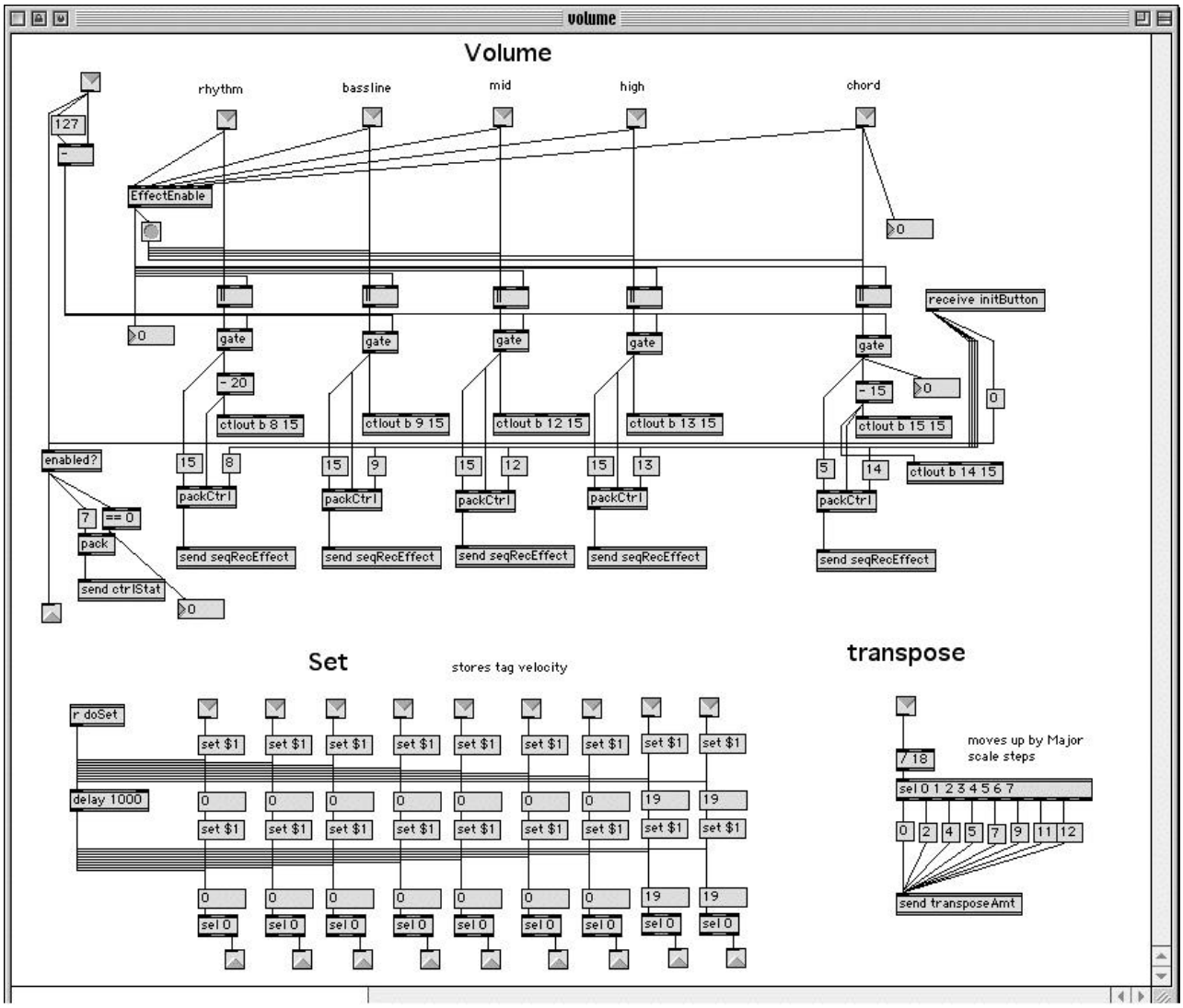
C.D Effects Patches

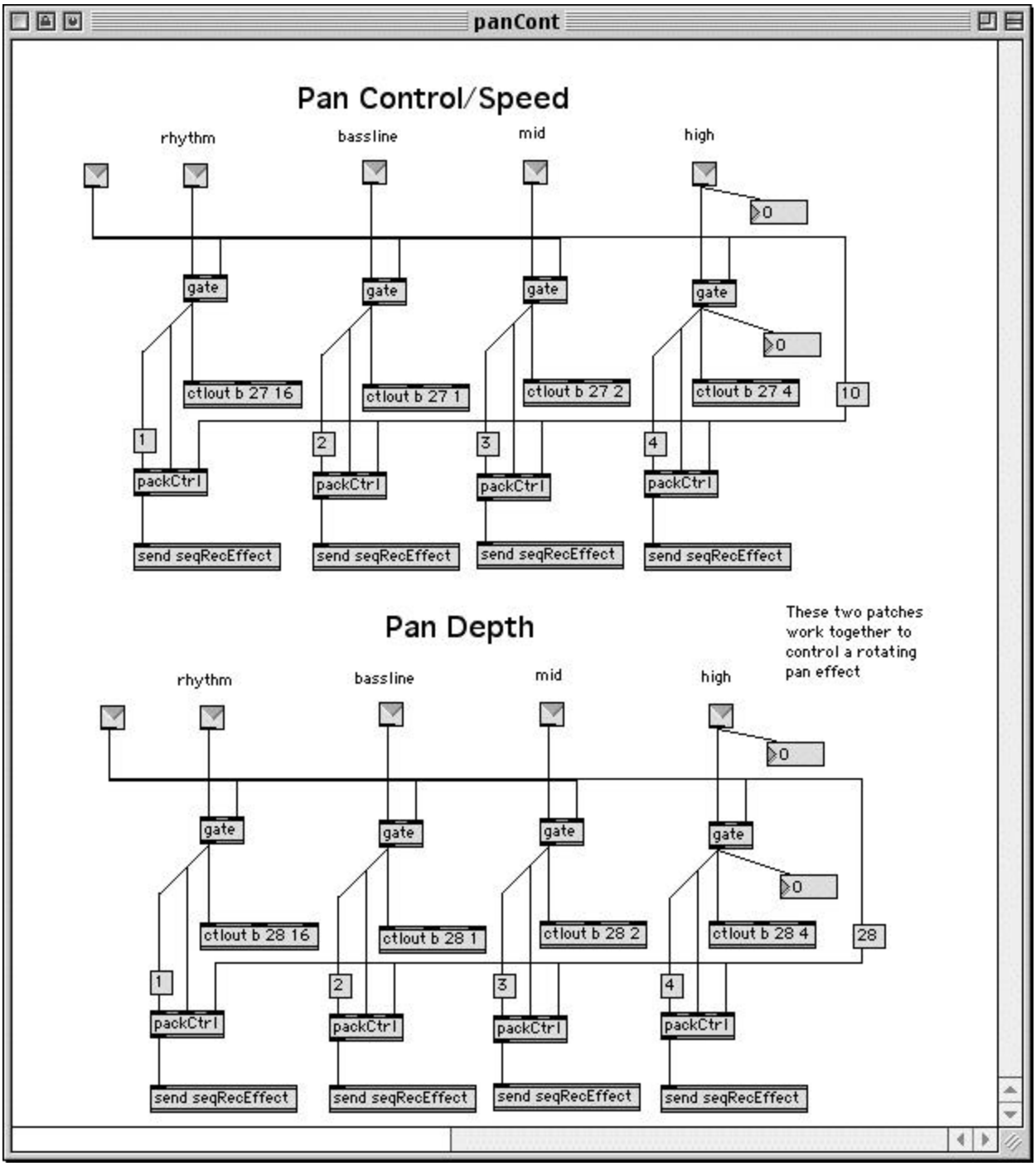


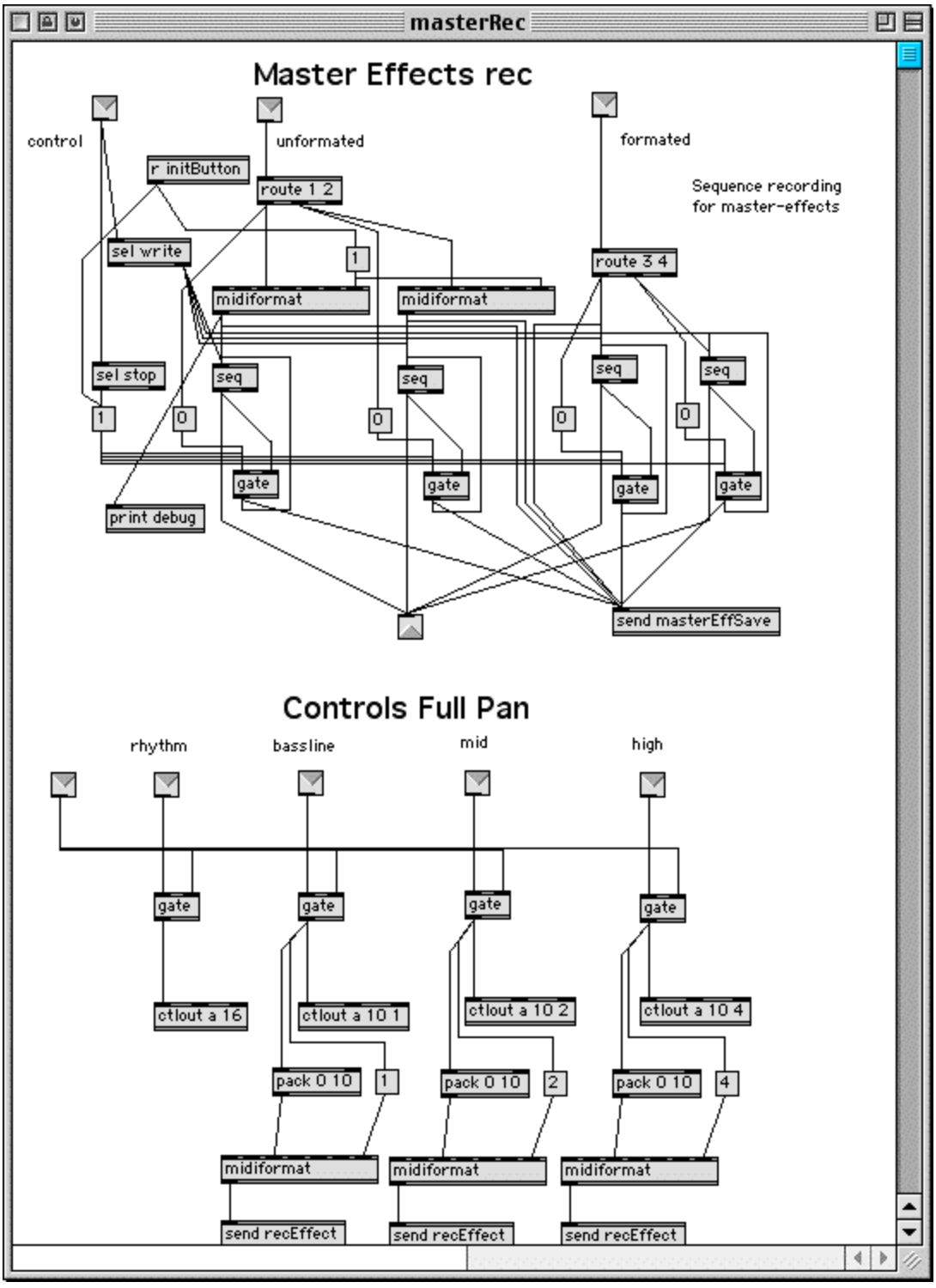

Resonance
chords
bassline
mid
high
r initButton
0
EffectEnable
>0
sel 0
4
80
30
100
gate
enabled?
== 0
71
pack
53
ctlout b 71 3
ctlout b 71 5
ctlout b 71 1
ctlout b 71 2
ctlout b 71 4
packCtrl
send seqRecEffect
ctlout b 53 3
ctlout b 53 5
ctlout b 53 1
ctlout b 53 2
ctlout b 53 4
send ctrlStat
this is used by rpgate to determine whether to screen the prior recorded control


picteff
Delay
presently running through channel 1 FXA
r initButton
14
0
127
pack 14 0
ctlout b 14 6
- 0
7 2
prepend 1
ctlout b 13 8
send mastereRecUData
>0
sends record to master effects seq
Distortion
presently running through channel 1 FXB
r initButton
0
pack 93 0
ctlout b 93 1
prepend 2
send mastereRecUData
sends record to it's own channel
Lexicon Control
axis 1
axis 2
xis 3
r initButton
1
10
7
pack 1 0
pack 10 0
pack 7 0
7 15
enabled?
enabled?
enabled?
16
midiformat
prepend 3
send newDemoLexicon
send mastereRecFData
switches Lexicon source allowing new app to use same setup as musical tinkets
click on which demo is running
MAX demo
musicaltrinkets demo
1
2
receive newDemoLexicon
midiin c
switch 2
gate
midiout c
print lexicon ctrl
Preset


eyeBall
Eyeball
axis 1
r initButton
ctlout b 25 1
sel 0
/ 2
23
ctlout b 26 1
ctlout b 24 1
enabled?
pack 24 0
pack 26 0
1
midiformat
prepend 4
send mastereRecFData
axis 2
ctlout b 25 2
sel 0
/ 2
63
ctlout b 26 2
ctlout b 24 2
enabled?
pack 24 0
pack 26 0
2
midiformat
prepend 4
send mastereRecFData
axis3
ctlout b 26 4
127
-
/ 2
ctlout b 24 4
enabled?
pack 24 0
pack 26 0
4
midiformat
prepend 4
send mastereRecFData
Effect Enable
r initButton
0
set $1
set $1
determines if no tags
present to determine
where line effects
are active or not
set $1
select 0
1
0


volume
Volume
rhythm
bassline
mid
high
chord
127
-
EffectEnable
0
receive initButton
gate
- 20
ctlout b 8 15
ctlout b 9 15
ctlout b 12 15
ctlout b 13 15
- 15
ctlout b 15 15
ctlout b 14 15
enabled?
15
8
9
12
13
5
14
packCtrl
send seqRecEffect
7
== 0
pack
send ctrlStat
Set
stores tag velocity
r doSet
set $1
delay 1000
0
19
sel 0
transpose
moves up by Major scale steps
% 18
sel 0 1 2 3 4 5 6 7
0 2 4 5 7 9 11 12
send transposeAmt


panCont
Pan Control/Speed
rhythm
bassline
mid
high
0
gate
ctlout b 27 16
ctlout b 27 1
ctlout b 27 2
ctlout b 27 4
10
1
2
3
4
packCtrl
send seqRecEffect
Pan Depth
These two patches work together to control a rotating pan effect
ctlout b 28 16
ctlout b 28 1
ctlout b 28 2
ctlout b 28 4
28


masterRec
Master Effects rec
control
unformated
formated
r initButton
route 1 2
Sequence recording
for master-effects
sel write
1
route 3 4
midiformat
midiformat
sel stop
seq
seq
seq
seq
1
0
0
0
0
gate
gate
gate
gate
print debug
send masterEffSave
Controls Full Pan
rhythm
bassline
mid
high
gate
gate
gate
gate
ctlout a 16
ctlout a 10 1
ctlout a 10 2
ctlout a 10 4
pack 0 10
1
pack 0 10
2
pack 0 10
4
midiformat
midiformat
midiformat
send recEffect
send recEffect
send recEffect

References

1. Hsiao, Kai-Yuh, Fast Multi-Axis Tracking of Magnetically-Resonant Passive Tags: Methods and Applications. February 2001. Available at, http://www.media.mit.edu/resenv/papers.html

2. Ishii, H. and Ullmer, B., "Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms, in *Proceedings of Conference on Human Factors in Computing Systems (CHI '97)*, (Atlanta, March 1997), ACM Press, pp. 234-241

3. Finkenzeller, K. *RFID Handbook,* John Wiley and Son, Ltd., New York 1998. pp. 62

4. Murakami, A., et al, "Position Detector," US Patent No. 5,466,896, Nov. 14, 1995.

5. Larry, O. "Marimba Lumina: This is not your mother's MIDI controller," *Electronic Musician*, 16(6). June 2000.

6. Ishii, H., Fletcher, R., Lee, J., Choo, S., Berzowska, J., Wisneski, C., Cano, C., Hernandez, A., and Bulthaup, C., "musicBottles", in *Conference Abstracts and Applications of SIGGRAPH '99*, Emerging Technologies, (Los Angeles, California USA, August 8-13, 1999), ACM Press, pp. 174.

7. http://tangible.media.mit.edu/projects/bottlogues/bottlogues.htm

8. http://tangible.media.mit.edu/projects/genieBottles/geniebottles.htm

9. Paradiso, J., Hsiao, K., Benbasat, A. Tangible Music Interfaces Using Passive Magnetic Tags, ACM CHI 2001 Conference - Special Workshop on New Interfaces for Musical Expression.

10. Hsiao, K., Paradiso, J. A New Continuous Multimodal Musical Controller Using Wireless Magnetic Tags, Proc. of the 1999 International Computer Music Conference, October 1999, pp. 24-27.

11. Paradiso, J. and K. Hsiao, K. Swept-Frequency, Magnetically-Coupled Resonant Tags for Realtime, Continuous, Multiparameter Control, Human Factors in Computing Systems; CHI99 Extended Abstracts, 1999, pp. 212-213.

12. Paradiso, J., Hsiao, K., Benbasat, A. *Musical Trinkets*: New Pieces to Play, SIGGRAPH 2000 Conference Abstracts and Applications, ACM Press, NY, July 2000, p. 90.

13. Downie, M. Behavior, Animation and Music: The Music and Movement of Synthetic Characters. M.S. Thesis, MIT Media Lab. January 2001

14. http://www.media.mit.edu/hyperins/rogus/

15. http://www.cycling74.com/

16. http://www.propellerheads.se/

17. http://www.korg.com/

18. EM Staff, "Most Innovative Product (Hardware/Software)" *Electronic Musician*, Jan 1 2000 EM staff

19. http://www.alesis.com/products/airfx/index.html

20. Rotondi, J. "ALESIS AirFX", *Remix*, Mar 1, 2001

21. Paradiso, J., Gershenfeld N., "Musical Applications of Electric Field Sensing", *Computer Music Journal.* 21(2), Summer 1997, pp. 69-89.

22. Glinksy, Albert. *Theremin: Ether Music and Espionage*, University of Illinois Press, 2000

23. Hasan, Leila. *Visual Frets for a Free-Gesture Musical Interface*, M.E. Thesis, MIT Media Lab. June 2001

24. Reason Operational Manual, 2000 pp. 145-154

25. Paradiso J., "Electronic Music Interfaces: New Ways to Play," *IEEE Spectrum Magazine*, Vol. 34, No. 12, pp.18-30 (Dec., 1997).