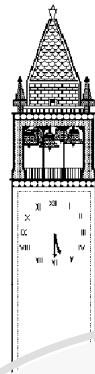
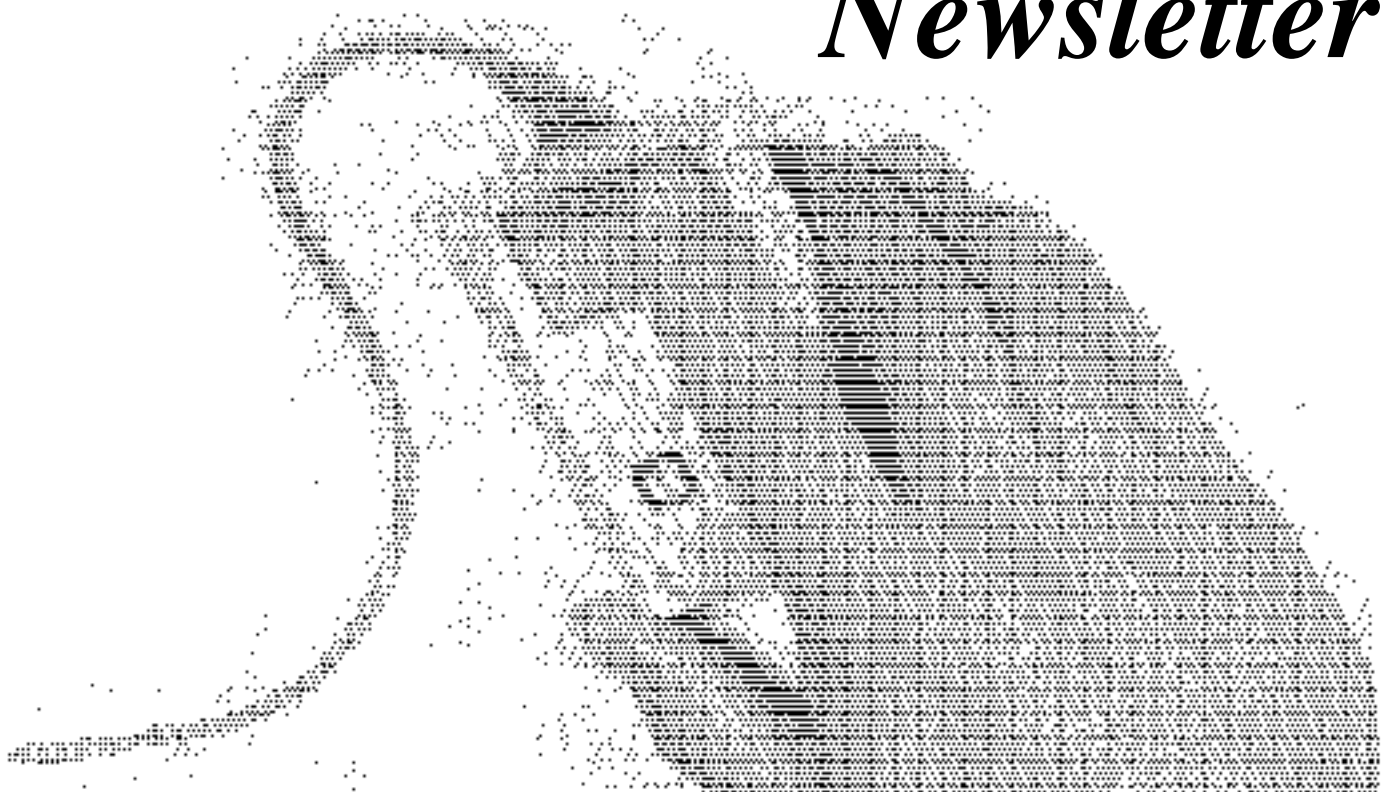


Berkeley Macintosh Users Group



***Fall 1985
Newsletter***



Official publication of the Berkeley Macintosh Users Group

Who We Are

BMUG is the Macintosh Users Group located at the University of California at Berkeley. This all-volunteer group is devoted to the exchange of information about the Apple Macintosh microcomputer and related products.

BMUG's membership consists of roughly 3000 people: students, faculty, and staff of U.C. Berkeley, Lawrence Berkeley and Livermore National Laboratories and other universities, as well as members from the surrounding community and all over the world.

MacRecorder Kit

BMUG offers a kit called MacRecorder II, which contains a speech digitizer and A/D converter. Cost is \$45 for the kit, which includes the circuit board, all parts, software, and instructions necessary to build the device. The microphone and a case are not included. The software comes as a double-clickable application; see "The MacRecorder Software" in this issue. MacForth source code is also included. See the articles by Ty Shipman and Michael Lamoreaux for more information.

MacFORTH Level 3

Creative Solutions, Inc., the company that designed MacFORTH, has kindly given BMUG a copy of their Level 3 upgrade. Level 3 allows programs developed in MacFORTH to be run without the full, proprietary MacFORTH kernel. Any BMUG member who has used their own copy of MacFORTH (Level 1 or 2) to develop an application that they would like to distribute as a **public domain (or shareware)** stand-alone application is welcome to contact Linda Custer at BMUG about using Level 3 to compile their application. If desired, we will provide help in converting the source code so that it is compatible with Level 3 and will work with you to produce a finished application.

In order to write code that will be compatible with Level 3, you should not require the end user to know or use any FORTH words. You should rely completely on the Macintosh interface for all input, since the MacFORTH dictionary is completely erased when processed through Level 3. Optimally, you should develop your application completely with Level 1 or 2, and then modify it only slightly for Level 3. Then use the resource editor or an icon editor to create icons and masks for your application and any files it creates. Lastly, create as many MacPaint screens as you want to show up as **About my application...** type menu items in the Apple menu of your application. Each screen should be exactly the size of the largest picture you can see at one time in MacPaint, and should be cut into a scrapbook file. Once you've gone that far, BMUG can show you how to do the rest.

If you have an application that is long and complex, requiring memory overlay capabilities at runtime, or other features of Level 3 with which we are not completely familiar, contact BMUG, and we may be able to work with you or lend you the documentation and software to see if you can work the details out yourself. We will ordinarily not let level 3 out to users.

Of course, if your application looks good when it's done, and if you would like, BMUG will be happy to add it to our public domain software collection (see above) and help you distribute it through various on-line services and software exchanges between user groups.

MacRecorder

A Speech digitizer for the Macintosh

by Ty Shipman

SMASH!!!! BOOoom!!!! OOF! RRoorrr....

These would be great sounds coming from the Macintosh in those great "run-time" games everyone is coming out with. Better yet, it would be great just to say **Open**, start talking, and have the Mac get the latest version of your life story you have been writing (hoping to make a million). At any point, why type? You could just speak into a microphone and have it become readable. Well, it's not too far into the future. Speech synthesis and voice recognition systems are hot in the electronics industry today. Did you realize that if you call your local information operator, a synthesized voice will say the number you wish—after a human tells the computer what number to say. But, in time, I think the person will be gone as well.

Back to those great sound effects you want in your next version of Kill the Enemy or Capture the Flag. Who wants to spend hundreds of hours (or dollars) trying to code those sounds? — NOT I. Well, with the help of the MacRecorder voice Digitizer, you can add them in only the time it takes to say SMASH!!!! BOOoom!!!! OOF! RRoorrr....

For a long time programmers, teachers, scientists, and many others have wanted to drive computers and their applications by voice and receive voice output instead of black and white dots on a 9-inch screen. MacRecorder allows you to do it. You can make the Macintosh respond through a voice command system and/or other general analog waves. You may also avoid those beeps telling you your disk is full. Instead, your computer will say "please change disks: this disk is full."

This project was developed to take sound (music, voice, or noise) and put it into a form the Mac can understand. Then digitizer can replay the sound, or any segment of it, utilizing the Mac's sound chip. You can also display the wave form on the screen.

HOW IT WORKS

The conversion of sound, which is an analog wave, to a digital sequence is called (you guessed it) analog to digital conversion (A to D). See Figures 1 and 2. A to D conversion is done by taking a reference point on the analog wave, usually the lowest point and calling it zero (0). The highest point on the wave is assigned a number, for example 255 (you'll see later why I choose that number). This assignment from high to low gives the computer the ability to recognize 256 different values.

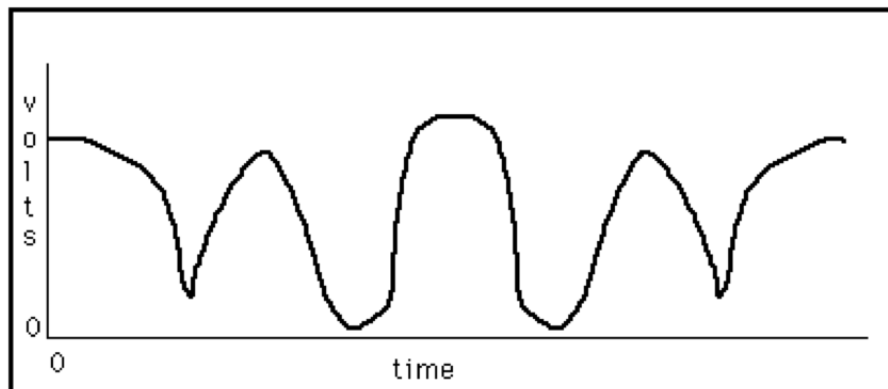


Figure 1: Simple analog wave starting at time zero.

When the computer sees a number like five (5), it sees a sequence of 1's (on) and 0's (off). Thus 5 to us looks like 00000101 (8 bit format) to the computer. This is known as a binary number, because it has only two values for each digit: one and zero.

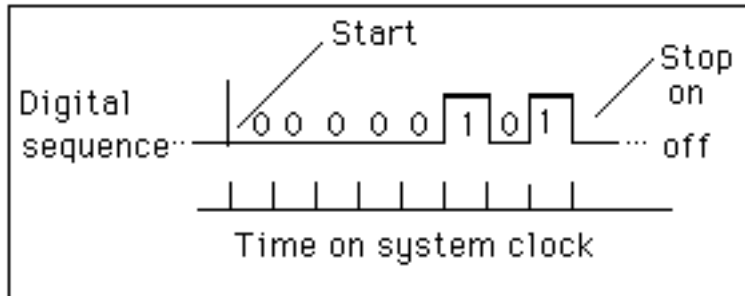


Figure 2: A digital sequence which represents the number 5 in binary (8-bit format)

Sampling an analog wave with a digitizer yields an intensity value consisting of a number ranging from 0 to 255 (Figure 3). With these numbers the computer can reconstruct both what the wave looked like, on the screen, and what it sounded like through the sound chip. The only limitation to a perfect reconstruction of the wave is the sample rate and the range of numbers that represent the intensity at the sample point. Essentially, the more samples taken, the better the reconstruction; the fewer samples, the poorer the reconstruction of the wave.

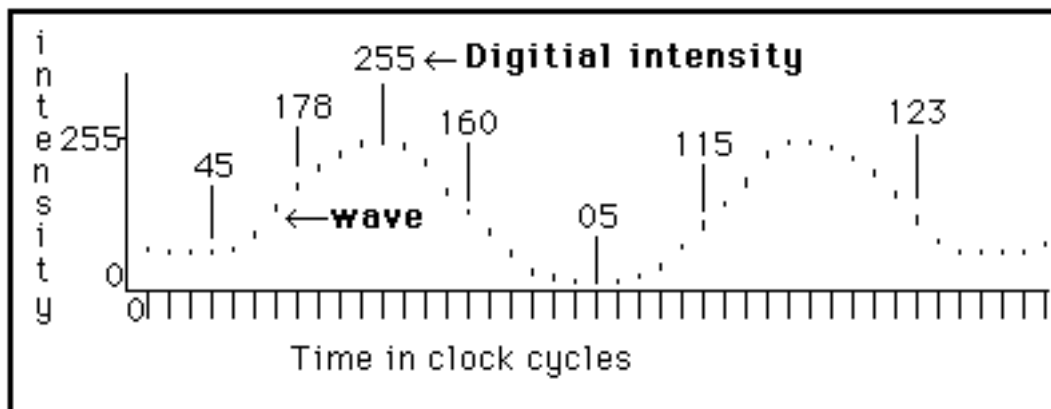


Figure 3: A sample of an analog wave showing digital conversions

Our project, using a sampling rate of 9321 samples per second and an 8 bit format (digital values 0 to 255) turn the Macintosh into a low fidelity tape recorder. This allows the Mac to reproduce sounds similar to those coming from your car's AM radio. The amount of time and quality that is recorded is very dependent on the speed of the software and memory size in your Macintosh. The 128K Mac with the current software records about 3 seconds of audio data. The 512K Macintosh records about 45 seconds of sound. The software is written in MacForth from Creative Solutions. A version written in C is planned.

THE HARDWARE

The MacRecorder is designed around an analog to digital converter made by National Semiconductor, the ADC0831 (Figure 4). This chip samples the sound wave intensity at V_{in+} and converts it into an 8-bit format (0-255). The digital output depends on the difference between V_{in-} (ground in this case) and V_{ref} (5 volts in this version). The A to D conversion starts when the chip select (**CS** U6;pin1) goes low. See Figure 5. This causes a start bit (low) to be sent out **D0** (U6;pin6). Following this start bit the eight data bits are sent, most significant first. The end of the transmission for that sample point is when the **CS** goes high, sending a stop bit (high). This is a forced high stop bit, which is accomplished by the pull up resistor

(R22). From **D0** the data is sent to the RS422 serial driver (U7;pin2), μ A9638c. This line driver is used to buffer both the data and the synchronizing signal, supplied by U4 pin 9.

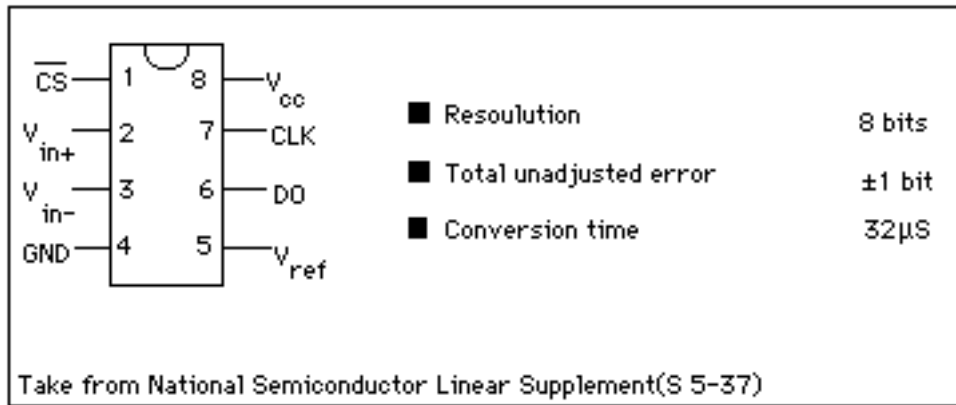


Figure 4: The heart of the MacRecorder

The data is sent to the Macintosh at a rate of 111,860 baud. This is so fast that we have also supplied a synchronizing signal to the Mac through pin 6;U7. The remaining circuitry is divided into two functions. The clocking portion (in Figure 5) is used to control the ADC. The portion used to preprocess the sound (in Figure 6) presents its output to the ADC (U6;pin 2).

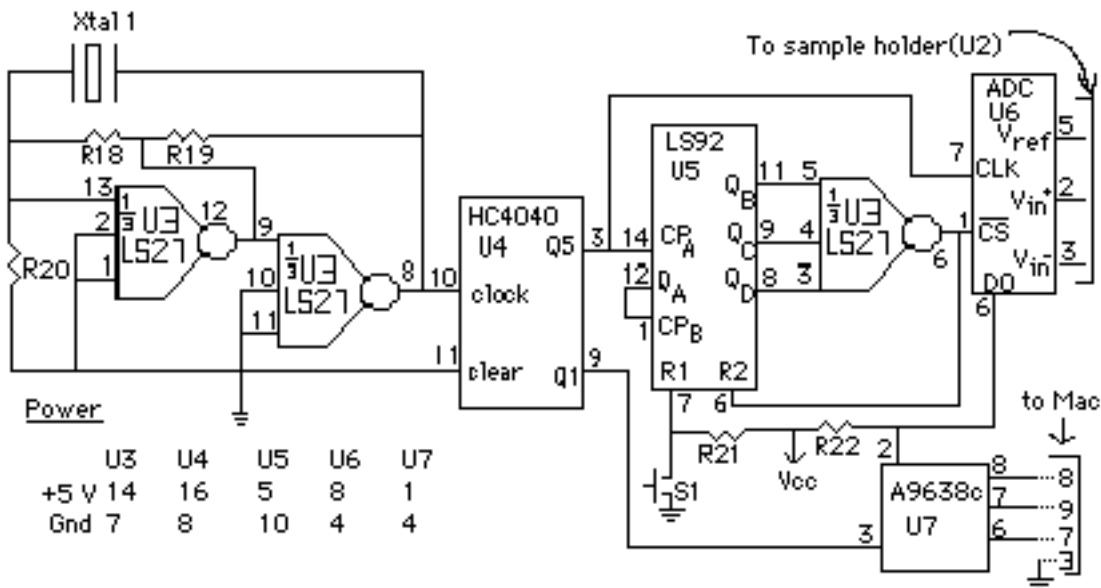


Figure 5: High speed serial A/D converter

The clock is made up of two NOR gates in the 74LS27 (U3) and a standard U.S. color burst crystal, 3.5795 MHz. This master clock frequency is divided by 2 in the 74HC4040 (U4) and output is at Q1 (pin 9;U4). This is the synchronizing signal that is used to drive the Serial Communications Controller (SCC) in the Mac (16 times the baud rate). Again the Master clock frequency is divided, this time by 32 with output through Q5 (pin3;U4). This signal is the actual transmission rate of the data. The signal at Q5 is also sent to the 74LS92 (U5), a divide-by-twelve counter. The output from this chip is sent to the last NOR gate on U3 which controls the **CS** line to the ADC. The actual **CS** line controls are as follows: two cycles high and ten cycles low. One low cycle is used for setup, another is for the start bit (low) and eight cycles are used for the eight data bits. One high cycle is used for the stop bit (high) and the other for setup. Thus the

sample rate at this speed is about 9321 samples per second or 74,568 bits per second.

The analog preprocessing (Figure 6) is done by the LM324 (U1), a quad op amp. The input is supplied by a standard Electret microphone and amplified by two op amps. Then the analog signal is sent to a third op amp that is configured as a low pass filter. This filter cuts out all frequencies above one-half the sample rate to prevent frequency aliasing. The high gain of the preprocessor is achieved by having the signal swing from 0 volts (V_{in-} is ground) to 5 volts (V_{ref}). Because of the swing from 0 to 5 volts the power supply to the LM324 must be 9 volts. This is a constraint of the LM324. The artificial ground for the voltage divider ($V_{ref}/2$) is supplied by the two 10K 1% resistors (R16, R17). From the low pass filter the signal is passed to an analog switch, the CD4066 (U2). This chip forms a sample and hold switch with the .022 μ F capacitor (C9).

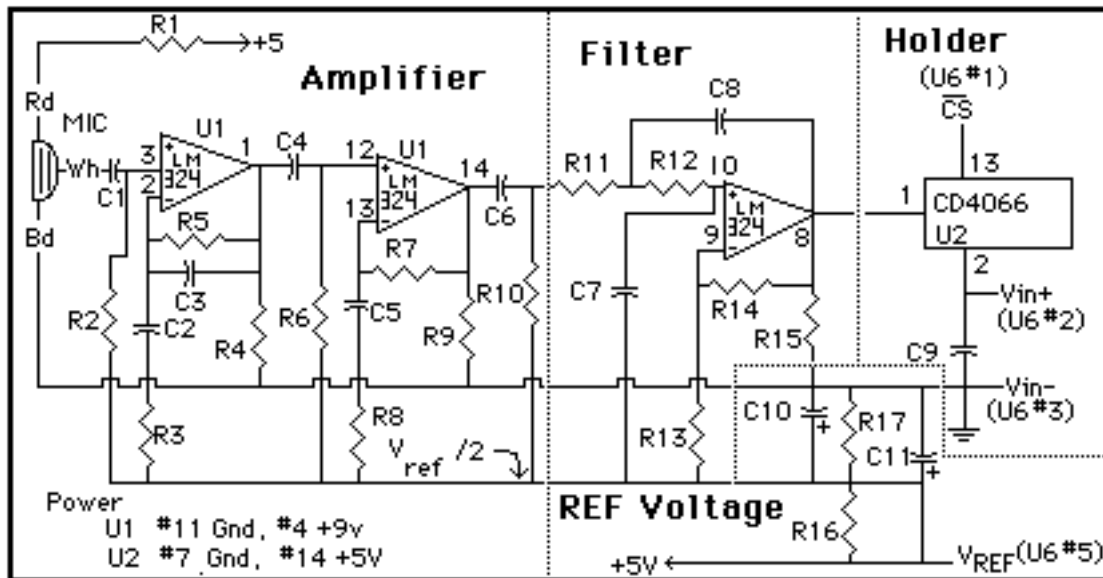


Figure 6: Amplifier, filter, and sample holder for voice digitizer

You might decide to replace all chips with their CMOS counterparts to run off the Mac's power. However, I advise against this! If you short out **any part** of the board, you may ruin your entire computer.

In case you were confused by of this "techie" stuff, a simple block diagram (Figure 7) explains it all. Really, it **is** this simple.

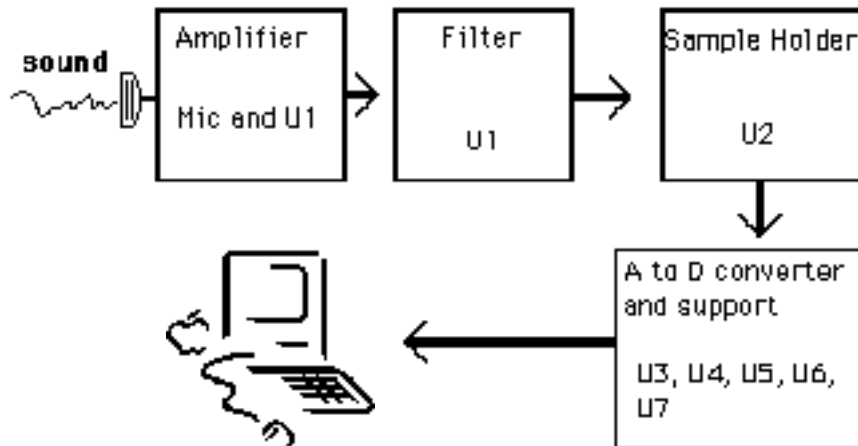


Figure 7: Block diagram of MacRecorder

The power supply (Figure 8) for the MacRecorder is a simple 9 volt DC wall transformer (P1), similar to a calculator power supply, plus some onboard regulation to supply 5 volts to the digital chips. A $220\mu\text{F}$ capacitor (C12) helps keep down those voltage ripples that would interfere with the A to D conversion. At this point, the line is split to supply the LM324 (U1) and the 5 volt regulator (U8), a 7805.

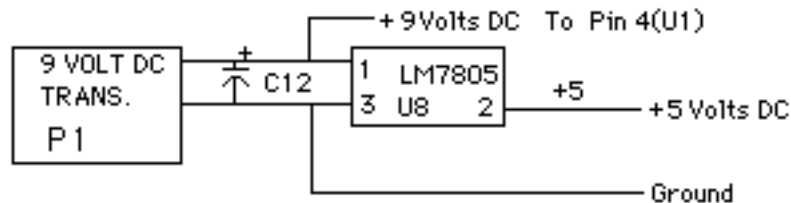


Figure 8: Power supply for MacRecorder

Parts list for MacRecorder

Resistors:

1/2 watt 5% carbon unless specified

R1, R21, R22	1 K Ω	R10,	10 K Ω
R2	5.6 K Ω	R11	11 K Ω
R3	1.1 K Ω	R12	6.2 K Ω
R4, R8, R9, R15	4.7 K Ω	R13, R20	22 K Ω
R5	220 K Ω	R14	68 K Ω
R6	100 K Ω	R16, R17	10 K Ω 1%
R7	91 K Ω	R18, R19	1.8 K Ω

Capacitors:

50 WVDC

C1, C7	0.01 μF	C9	0.022 μF
C2, C4, C5, C6	0.1 μF	C10, C11	100 μF 25V
C3	200 pF	C12	220 μF 25V
C8	0.0033 μF		

Fall 1985 BMUG Newsletter

Semiconductors:

U1	LM324	U5	74LS92
U2	CD4066	U6	ADC0831
U3	74LS27	U7	μ A9638c
U4	74HC4040	U8	LM7805

Miscellaneous:

Xtal1	U.S. color burst crystal 3.5795 MHz
P.C. board	Available from BMUG
P1	9 volt DC power supply (at least 300mA)
Mic	Electret Microphone (Radio Shack #270-092) or equiv.
S1	Normally open momentary contact Switch
S2	STSP toggle switch (not shown power on/off)
Connectors	Power supply type, DB9 Male (better Joy Stick extender by Radio Shack), Microphone connector(s) if your microphone is in a housing
IC Sockets	2 eight pin, 1 sixteen pin, 4 fourteen pin
Cabinet	at least 20cm(L) x 14cm(W) x 8cm(H)
Supplies	solder, iron, cutters, wire (for jumpers),...etc.

Other ideas for the MacRecorder

If you're the adventurous type you may want to hook up straight into the ADC0831. Here is how. You must put 10K Ω pots on both the V_{in-} and the V_{ref} (Figure 9). You should not let V_{in+} fall below 1 volt. If this happens the ADC will incur a lot more error than the ± 1 bit. You will have to amplify the input through an op amp. The reprint of the data contains ideas on how to hook up sensors to the ADC0831.

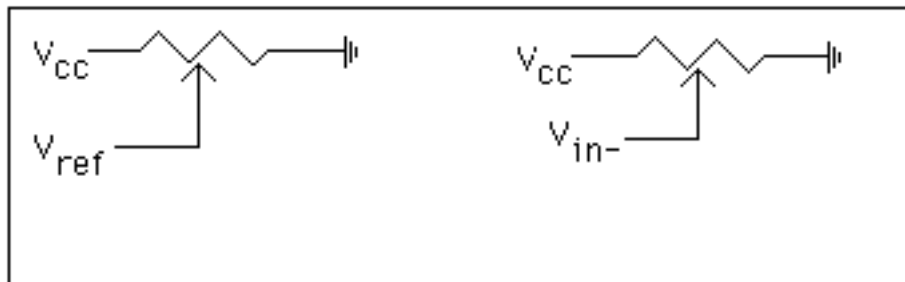
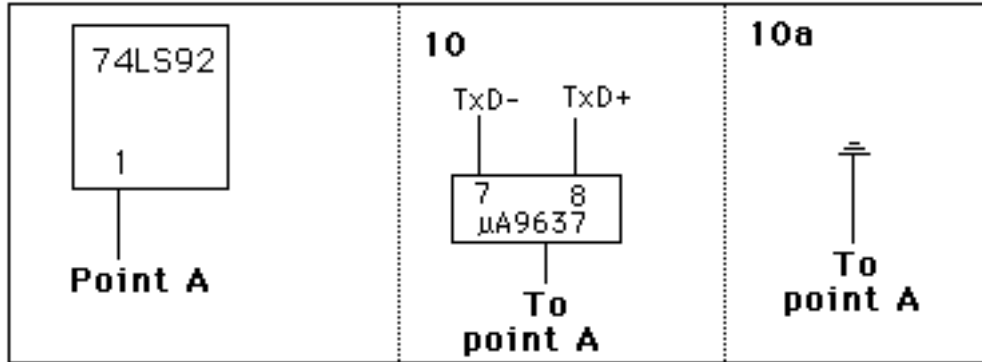


Figure 9: Hooking 10K Ω pots into ADC0831 for adjustable analog inputs

Also, if you want remote control over the sending portion of the system, do it through the software, taking only the sample you want. If you're really hard up you can add an μ A9637, which is an RS-422 receiver. See Figure 10 for the exact procedure. Then you just send a 00H (hex) out the serial port and the receiver will do the rest. I must mention that this hasn't yet been tried, so you may have to experiment. To set up continuous data transmission, see Figure 10a.



Figures 10 and 10a: Remote polling and continuous polling of MacRecorder

Another improvement on the circuit is to replace the crystal with one of a higher value, such as 4.9152 MHz. This increases the sample rate to 12800 samples per second. You would also have to modify the software to handle the increased speed (modify Snd.rate in the FORTH version). Figure 11 is a list of the pinout for the Macintosh's serial port. If you decide to modify the circuit for auto polling you'll need it.

M A C	1	2	3	4	5	6	7	8	9
U S E	GND	GND		TXD-	HSK			RXD-	
	+5V		TXD+	+12V				RXD+	

HSK = sync line for external timing

Figure 11: Macintosh RS-422 serial port configuration

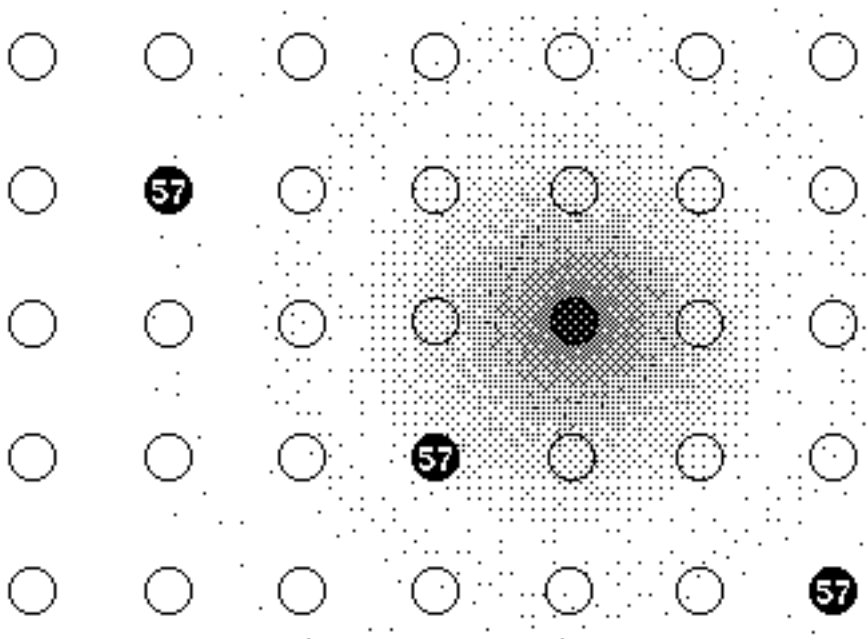
Conclusion

If you are a nontechnical person, this kit is for you too. It is not hard to build. With its step-by-step construction book, you need only a steady hand and some patience. I encourage all who want to experiment with a voice recognition system to buy the kit and build it. Once again: no technical knowledge of electronics is necessary.

The MacRecorder expands the range of applications for the Mac. One could modify the operating system to take input from the MacRecorder. The MacRecorder is also an option for disabled people currently not benefitting from the computer revolution because it is too time consuming to type all the commands or to control the mouse.

The original designer of the MacRecorder is Michael Lamoureux. Michael is currently at the University California, Berkeley working towards a Ph.D. in mathematics.

Michael and I are designing a full 4 or 8 channel A to D converter, along with a 4 or 8 channel D to A converter with controls. Look for it in the Spring '86 BMUG Newsletter.



Ruderman-Kittel-Kasuyda-Yosida Electron Spin Density Disturbance

Brent Fultz

MacRecorder II

An improved voice digitizer for the Macintosh

© 1985 by Michael Lamoureux

Warning: This article is unabashedly technical. Briefly, it describes an improved version of the MacRecorder voice digitizer that was introduced in Ty Shipman's article. This circuit is simpler than the one Ty described, with added features such as volume control and improved noise characteristics. By the time you read this, BMUG should be offering a complete kit for the MacRecorder II. At this point, the software for the MacRecorder turns your Mac into a two thousand dollar audio cassette recorder. (That's \$2,000 for the Mac, not the kit!) We're hoping that by distributing this inexpensive circuit, other applications will be developed, such as voice recognition systems, video digitizers, specialized lab equipment and so forth. Moreover, by using this as a standard interface for A to D conversion, wide distribution of many applications is possible.

Introduction

The MacRecorder voice digitizer described in the last article by Ty Shipman is in fact just one circuit in a long series of digitizers I've built for the Macintosh. That particular version has a number of problems, most notably the need for a dual power supply (i.e. 9 volts to the op amp chip, and 5 volts to the digital chips), the use of a low performance sample/hold circuit (the 4066 with capacitor), and the overall complexity of the circuit. In addition, the low pass filter circuit cuts off useful frequencies while the A to D converter chip (the ADC0831) introduces noise to the sampled waveform due to its "total sampling error" of 1 significant bit. (Total sampling error is a characteristic of the converter chip, and can be thought of as random noise.) Nevertheless, it is a useful and instructive circuit with remarkably good fidelity.

However, I couldn't let a good thing sit, so I redesigned the MacRecorder to come up with the MacRecorder II. This circuit is based on a Texas Instruments chip, the TLC549, which is an 8 bit serial A to D converter, similar to the ADC0831 used in the original MacRecorder. However, the TI chip has a total sampling error of only .5 significant bits, which means less noise. It also has a built-in sample/hold circuit which eliminates the 4066 chip from the circuit. Moreover, the microphone amplifier has been redesigned to include a gain adjustment (i.e. volume control) and the filter now has the right frequency cutoff near 4800 Hz, that is, at one-half the sample rate of 9600Hz. Looking ahead to other applications, this circuit can easily be upgraded to a faster sampling rate simply by changing the crystal. In fact, I've tested it at a sampling rate of 19200Hz and found no problems.

By the way, if you haven't read Ty's article on the MacRecorder yet, I recommend you do so now, as I won't repeat here the basics of A to D conversion and the MacRecorder hardware which he covers very well.

About The Circuit

Figure 1 shows the digital part of the MacRecorder II. The 74HC4060 is a 14 stage binary counter and oscillator which provides the master clock signals for the circuit. Pin 9 of the 74HC4060 sends a clock signal to the Mac serial port through the μ A9638 line driver (i.e. buffer). The 74HC20 generates "start bits" and "stop bits" for producing the standard asynchronous communications format to the Mac's serial port. The TLC549 is the A to D converter which accepts analog data at pin 2 (Analog In), and outputs binary data at pin 6 (D0), which is buffered by the μ A9638 and sent to the Mac. R4 and R5 are 10 turn potentiometers (variable resistors) that are adjusted to set the reference voltages Ref+ and Ref- on the TLC549. For proper operation of the circuit, R4 must be adjusted so that Ref- is at 1.5 volts, and R5 adjusted so that Ref+ is at 3.5 volts. (The TLC549 measures the Analog In voltage relative to these two reference voltages. The values mentioned are ones I've found useful.)

Figure 2 describes the analog portion of the voice digitizer. Resistors R7 and R8 form a voltage divider to give an artificial ground for the op amps. The first op amp is a high gain amplifier which boosts the signal from the microphone. Note that the potentiometer R11 is the gain control, which is adjusted to set the

sensitivity of the microphone (to set for recording either whispers or only shouts). The second op amp is configured into a low pass two pole Chebyshev filter with a cutoff frequency of 4660 Hz and a pass band ripple width of .5 dB. This stage only passes sound with frequencies less than roughly one-half the sample rate while blocking higher frequencies. The .5 dB PRW is a measure of how uniformly this stage passes the lower frequencies. (.5 dB is pretty good.)

One thing missing on the MacRecorder II is a push button to start the conversion process. In fact, the circuit is continuously sending information to the Macintosh, and it is the job of the software to accept or ignore the data. The advantage of this method is that the hardware is now under software control, which means everything can be controlled with the mouse: user friendly!

By the way, this circuit is easily adapted to digitizing other voltage sources. Just use the circuit in Figure 1, and send the voltage to be sampled to Pin2 on the TLC549 (Analog In). Use R4 and R5 to set Ref- and Ref+ to the minimum and maximum voltage that you expect to measure on Pin2. Keep in mind that Ref- and Ref+ must lie between Gnd and Vcc (5 volts), and for accuracy in the conversion, Ref+ should be at least one volt greater than Ref-.

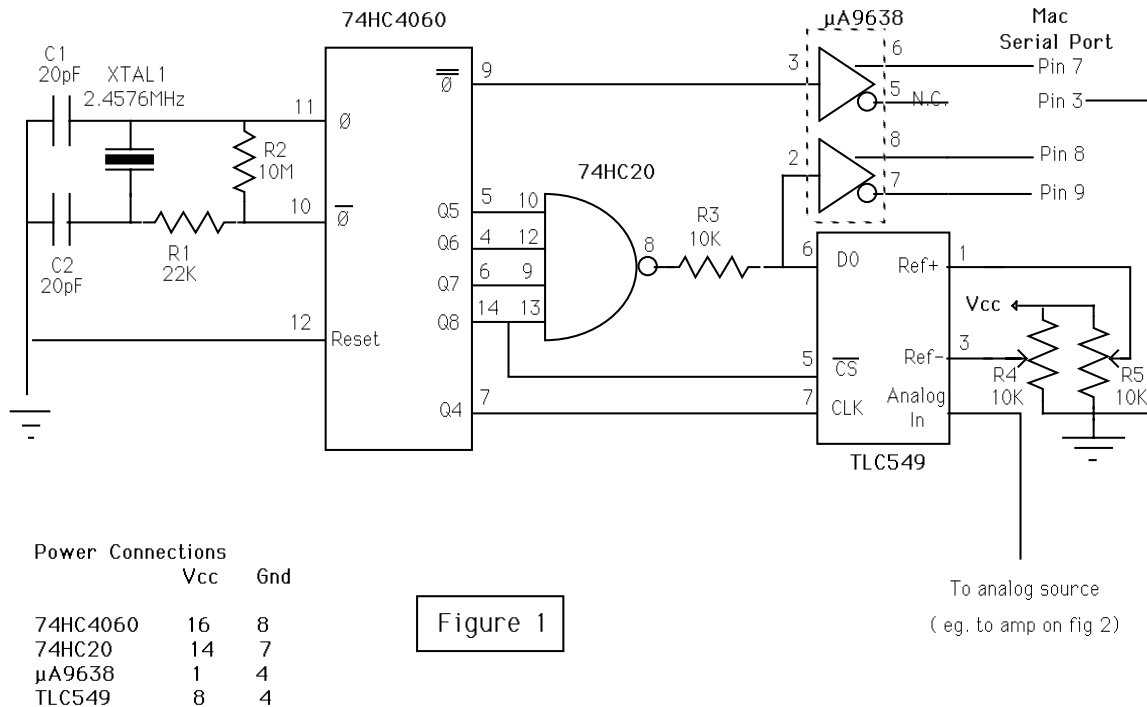
Finally, in case you wish to try other sample rates, note that we have the formula:

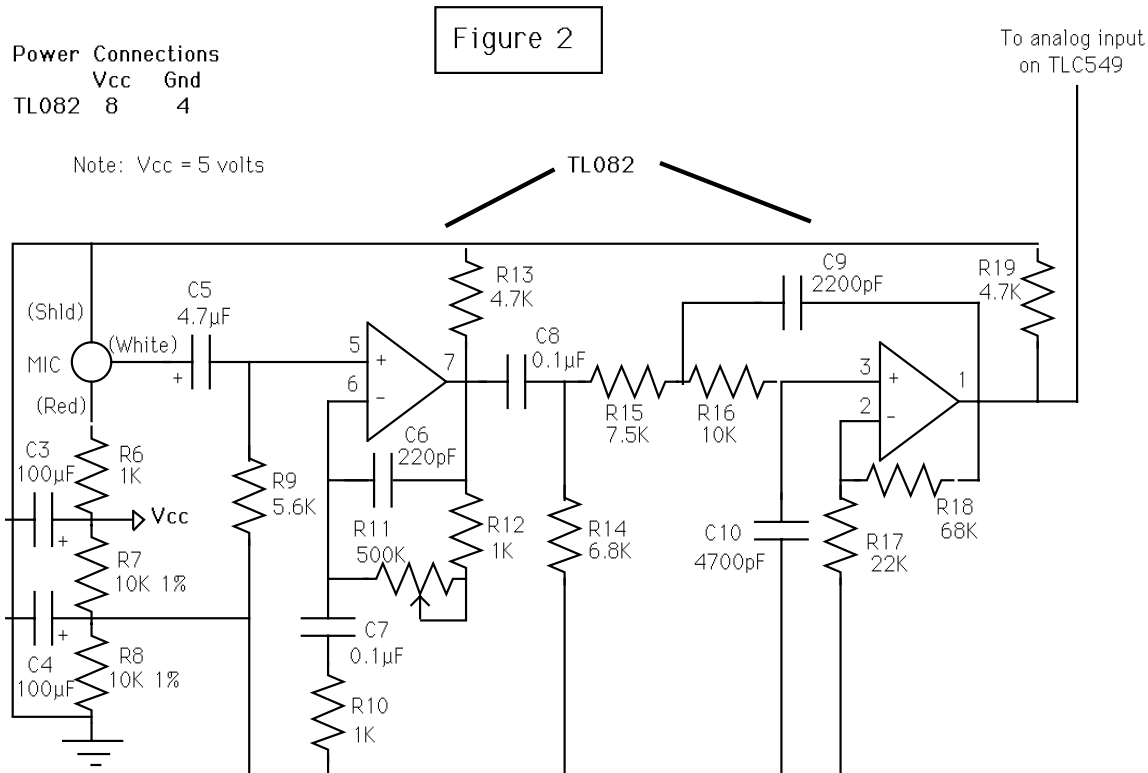
$$\text{SAMPLE RATE} = \text{CRYSTAL FREQUENCY} / 256$$

and so by simply changing the crystal, you can change the sample rate. However, the data sheets for the TLC549 suggest that you can't expect it to work at sample rates over 29000Hz, while the 74HC4060 may not oscillate reliably with crystal frequencies much over 4MHz.

Circuit Availability

BMUG offers the MacRecorder II as a kit, which includes a printed circuit board, power supply and cables to connect it to your Mac. It costs \$45, including the "cassette recorder" software. If you have any questions or suggestions about the circuit, you can easily contact me by mail (care of BMUG), or send messages to my electronic mailbox in the BMUG BBS. In later newsletters we will cover further applications of the MacRecorder.





Parts List for the MacRecorder II

Capacitors (all at least 9 WVDC)

C1,C2	20 pF
*C3,C4	100 µF electrolytic
*C5	4.7 µF electrolytic
*C6	220 pF
*C7,C8	0.1 µF
*C9	2200 pF (use a good quality Mylar or similar)
*C10	4700 pF (ditto)

Integrated Circuits

74HC4060	14 stage binary counter with oscillator
74HC20	Dual 4 input NAND gate
µA9638	Dual RS-422 line driver
TLC549	8 bit serial A-D converter (Texas Instruments)
*TL082	Dual biFET op amp
7805	5-volt power regulator (not shown)

Resistors (all 1/4 watt, 5% unless otherwise specified)

R1,R17	22K
R2	10M
R3	10K
R4,R5	10K, 10-turn potentiometer
*R6,R10,R12	1K
*R7,R8	10K, 1% precision resistor
*R9	5.6K
*R11	500K potentiometer
*R13,R19	4.7K
*R14	6.8K
*R15	7.5K
*R16	10K
*R18	68K

Miscellaneous

XTAL1	2.4576 MHz crystal
*MIC	Radio Shack #270-092 Electret condenser mike element
DB-9 connector	(male)

Note: parts with a * are those used only in the microphone amp and filter (Figure 2).

The MacRecorder Software

by Michael Lamoureux

Introduction

The MacRecorder II is a hardware device used to digitize voice waveforms and pass them to the Macintosh through the serial port. The software that accompanies the MacRecorder II basically turns your Macintosh into a cassette tape recorder, allowing you to record sounds, play them back and save the recordings to disk. In addition, some editing features are provided, to allow you to "doctor" the recording by adding or deleting sounds, or rearranging their order, in a manner similar to MacWrite. This concept of the Mac as a mixed tape recorder/word processor will make the controls more intuitive to use.

Description

I'll begin by describing the typical Macintosh screen displayed by the MacRecorder, Figure 1 below. This screen was reached by doubling clicking on the MacRecorder icon, and opening (using the FILE menu) a data file called Lamoureux.9600. There is one window on the screen, which is given the title of the file opened. The most prominent feature of the window is the graphics display, which is an amplitude verses time representation of the voice waveform stored in the opened file. The horizontal axis is the time axis, while vertically is the amplitude of the wave at each instant. Usually the stored waveform is bigger than what can fit on the screen, so the scroll bar along the bottom of the window is used to scroll along the entire waveform, just as one scrolls through a document in MacWrite. Above the graphics area are five controls. **Play**, **Record** and **Stop** work exactly as you would expect them to (more on **Record** later), while **RePitch** and the sliding control next to it are used to adjust the playback speed of the recording, just like a speed control on a tape recorder. **RePitch** resets the sliding pitch control to the normal setting.

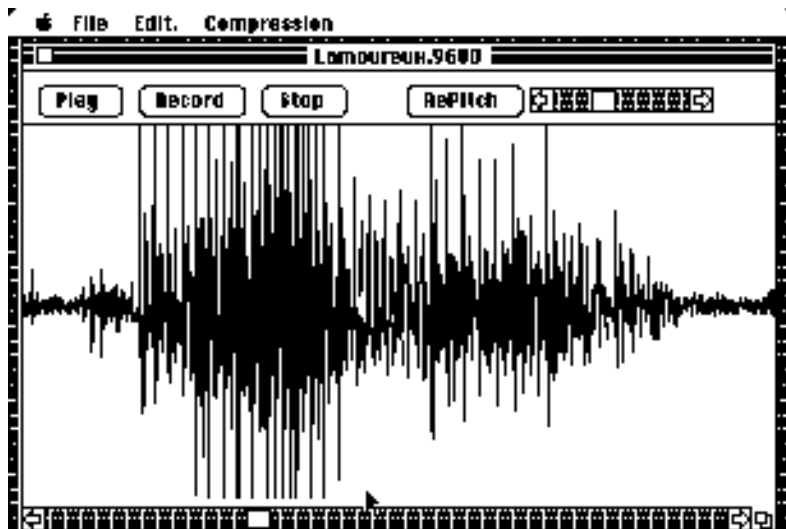


Figure 1

A user can select a portion of the waveform by clicking and dragging with the mouse, just as one selects text with MacWrite. Of course, the waveform scrolls along as you select by dragging off the screen, while mouse clicks with the shift key held down extends the region of selection. Figure 2 shows a selected region of the waveform, as indicated by highlighting. Pushing the **Play** button now would play only the selected region. Pushing **Record** would record over the selected region, erasing it in the process. If no region is selected, the insertion point for the next **Record**, or beginning point for playback is indicated by a vertical bar on the screen.

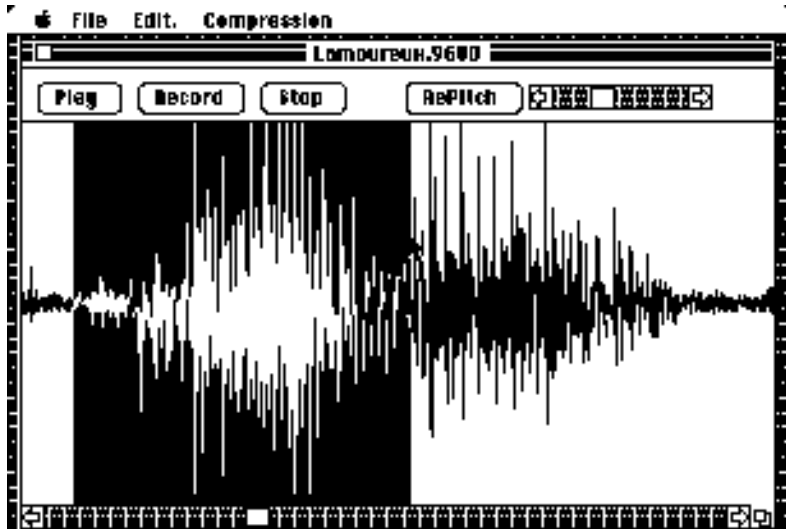


Figure 2

The **Record** button has some special features now which allow the user to set the total length of recording time. When you hit **Record**, a new window opens, titled **Recording**, as shown in Figure 3. On it are two push buttons, **Start** and **Stop**, a message indicating the total length of sound to be recorded, and a sliding control for you to set this length. (On a 128K Mac, the maximum recording time is just over 3 seconds, while the 512K ("Fat") Mac can record over 40 seconds worth of sound.) Once the recording length is set, you hit **Start** to begin the actual recording. Thus, the **Record** button only sets up the machine for recording, while the **Start** button actually "gets the tape rolling".

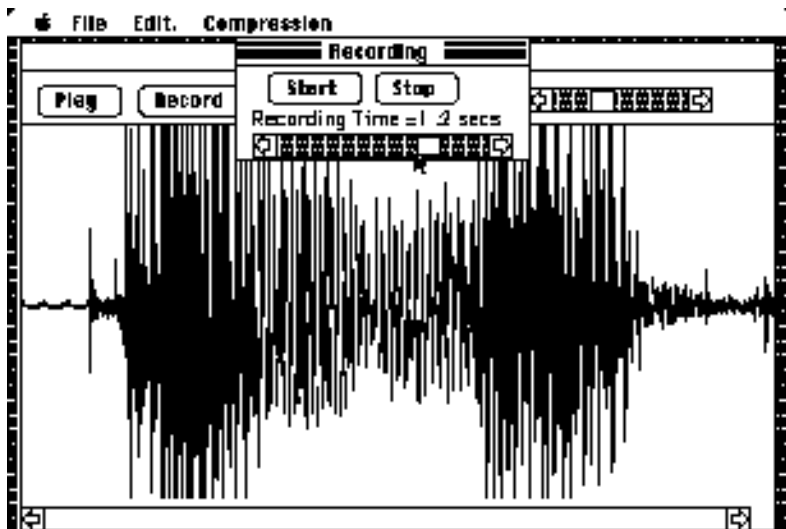


Figure 3

Of the four menus at the top of the screen, three are familiar. The **Apple** menu contains the usual desk accessories, plus some information about the program. **File** allows you to save and retrieve waveforms on the disk, and to quit the program. **Edit** provides for cutting and pasting operations. Under the **Edit** menu is also a **Reverse** item, which reverses the order of a selected waveform, so at playback, the sound comes out backwards (my brother's suggestion, and it's fun). The fourth menu is the **Compression** menu, which simply formats the graphics display in order to condense the waveform so that more of it fits on the screen. Note that **Compression** does nothing to change the sound of the waveform. Think of it as changing the font size in MacWrite, so that more of the document can be seen on the screen.

Using the Software

The easiest way to become familiar with the software is to open a data file that is already on the disk, and experiment with **Playing** the waveform, scrolling along it, selecting and **Reversing** sections, **Clearing** out selected sections, and adjusting the pitch. You don't need to have the hardware installed to do this, and since this is Shareware, you are free to use it to send and receive spoken messages to and from other people.

To use the **Record** command, you must have the MacRecorder hardware. First, close any file you now have open and select **New** under the **File** menu, to start fresh. Attach the 9 pin plug from the MacRecorder to the modem port on the Mac, and plug in the power supply for the MacRecorder. Hit **Record**, and then under the **Recording** window, set the **Recording Time** to a few seconds by sliding the thumb on the slide control. Then hit **Start**, and speak into the MacRecorder microphone. The **Recording** window disappears when the recording is done, and the newly entered waveform appears on the screen, highlighted to show it is the selected waveform. Now hit **Play** to hear what you said. At this point, you can edit the waveform, save it to disk, or add more sound by hitting **Record** again (but keep in mind that a selected region is erased and recorded over. Think about how MacWrite replaces selected regions with typed-in text and you will see why this program records over selected regions).

General Problems

Since the whole waveform you wish to play must be kept in memory, the amount of memory in your Mac is a big restriction. It doesn't take long to fill up all the memory at this recorder's 10K Bytes per second. On a 128K Mac, 3 seconds can be recorded, which is about a sentence, while on a Fat Mac, the 40 seconds you have amounts to a lot of talking. Doesn't this remind you of good old MacWrite version 1.0?

Program Listing

The following pages are listings of the MacRecorder software for reference. The software comes with the kits, and is also available on BMUG disk #4 (MacForth Disk). The listing is the latest version as of this writing — 0.8 — which doesn't implement all editing features and has a few problems.

Conclusion

At this point, a software reviewer usually tries to give an objective summary on the good and bad points of a product, and a rough idea of how useful it is. However, as the designer of both the software and the hardware for the MacRecorder, I don't feel the least bit objective. However, I do know that at both the MacWorld Expo in Boston this summer, and the PC Faire in San Francisco this fall, people were very excited about this "Macintosh tape recorder". I also know it's nice having your own digitizer in your home, especially if you have any interest at all in linguistics, voice recognition or voice synthesis, and this device is a convenient replacement for the minicomputers usually used for those jobs. Games programmers will find it useful in adding special sound effects to their software. Computer dating services could use it to keep a sound recording of prospective partners! Probably the most useful thing I've found is that the **MacRecorder** has made me very aware of how I speak, letting me hear right away which syllables I drop, how my voice sounds when I'm tired, and so on. I'm sure you can think of applications too: as another Mac speech program concluded in its demo, "just imagine ... the possibilities!"

For information on the MacRecorder hardware and software,
contact:

Michael Lamoureux
3024 A Fulton St.
Berkeley, CA 94705

or c/o the Berkeley Macintosh Users Group

Copyright © 1985 by Michael Lamoureux

All rights reserved.

(Copyright 1985 by Michael Lamoureux)

```
INCLUDE" ADC1"
: fopening      ( -- | dimming unuseable menu items )
    1 false file.menu.id item.enable
    2 false file.menu.id item.enable
    3 true  file.menu.id item.enable
    4 true  file.menu.id item.enable
    5 true  file.menu.id item.enable ;
: fclosing      ( -- | dimming unuseable menu items )
    1 true  file.menu.id item.enable
    2 true  file.menu.id item.enable
    3 false file.menu.id item.enable
    4 false file.menu.id item.enable
    5 false file.menu.id item.enable ;
cr ." Including windows and controls"
35 CONTROLS.VSIZE !                ( vert size of control region )
NEW.CONTROL PLAY.BUTTON             ( play button control )
    A.PUSH.BUTTON PLAY.BUTTON C.TYPE
    " Play "          PLAY.BUTTON C.TITLE
    10 10             PLAY.BUTTON C.POSITION
NEW.CONTROL RECORD.BUTTON           ( record button control )
    A.PUSH.BUTTON RECORD.BUTTON C.TYPE
    " Record "        RECORD.BUTTON C.TITLE
    76 10             RECORD.BUTTON C.POSITION
NEW.CONTROL STOP.BUTTON             ( stop button control )
    A.PUSH.BUTTON STOP.BUTTON C.TYPE
    " Stop "          STOP.BUTTON C.TITLE
    156 10            STOP.BUTTON C.POSITION
NEW.CONTROL REPITCH.BUTTON          ( repitch button control )
    A.PUSH.BUTTON REPITCH.BUTTON C.TYPE
    " RePitch "       REPITCH.BUTTON C.TITLE
    250 10            REPITCH.BUTTON C.POSITION
NEW.CONTROL PITCH.CONTROL           ( pitch button scrolling
                                   control )
    A.SCROLL.BAR      PITCH.CONTROL C.TYPE
    12 330 28 450     PITCH.CONTROL C.BOUNDS
    100               PITCH.CONTROL C.VALUE
    50 200            PITCH.CONTROL C.RANGE
NEW.WINDOW RECORDING.WINDOW         ( setup recording window )
    " Recording "      RECORDING.WINDOW W.TITLE
    37 150 97 340     RECORDING.WINDOW W.BOUNDS
    NOT.VISIBLE       RECORDING.WINDOW W.ATTRIBUTES
NEW.CONTROL START.BUTTON            ( start button definition )
    A.PUSH.BUTTON     START.BUTTON C.TYPE
    " Start "         START.BUTTON C.TITLE
    20 5              START.BUTTON C.POSITION
NEW.CONTROL RSTOP.BUTTON            ( recording stop button
                                   definition )
    A.PUSH.BUTTON     RSTOP.BUTTON C.TYPE
    " Stop "          RSTOP.BUTTON C.TITLE
    90 5              RSTOP.BUTTON C.POSITION
NEW.CONTROL TIME.CONTROL            ( set recording time scroll
                                   control )
    A.SCROLL.BAR      TIME.CONTROL C.TYPE
    44 10 60 180      TIME.CONTROL C.BOUNDS
    0                 TIME.CONTROL C.VALUE
```

```

0 0          TIME.CONTROL C.RANGE
: PLAYBACK. ( -- | plays between the margins, with error
               checks )
    LT.MAR @ RT.MAR @ OVER - DUP 0>
    IF PLAY.BYTES          ( play if rt.mar > lt.mar )
    ELSE DROP RCD.DSIZE @ OVER - DUP 0>
        IF PLAY.BYTES          ( else play if dsize > lt.mar )
        ELSE 2DROP
    THEN
    THEN ;
: RECORD. ( -- | deletes selected region, calls up
            RECORDING.WINDOW )
    TRUE changes ! fopening
    LT.MAR @ RT.MAR @ OVER - DUP 0>
    IF RCD.DELETE DROP LT.MAR @ RT.MAR ! ( delete region )
    THEN RECORDING.WINDOW SELECT.WINDOW ;
: @PITCH.CONTROL ( -- current value | checking pitch control )
    PITCH.CONTROL @ GET.CONTROL ;
: !PITCH.CONTROL ( value -- | set pitch control, and snd.rate )
    50 MAX 200 MIN DUP SAMPLE.RATE @ * 100 / 65536 * 22257 /
    SND.RATE !
    PITCH.CONTROL @ SWAP SET.CONTROL ;
: INC.PITCH.CONTROL ( increment -- | increments pitch control )
    @PITCH.CONTROL + !PITCH.CONTROL ( inc control )
    LAST.PART UPDATE.CONTROL          ( highlight the part )
    0 UPDATE.CONTROL ;                ( unhighlight the part )
: DO.PITCH.CONTROL ( increment -- | handles mouse to
                    increment pitch )

    BEGIN MOUSE.BUTTON
    WHILE DUP INC.PITCH.CONTROL
    REPEAT DROP ;
: TRACK.PITCH.THUMB ( -- | tracks thumb on pitch scroll control )
    FOLLOW.MOUSE DROP
    @PITCH.CONTROL SAMPLE.RATE @ * 100 / 65536 * 22257 /
    SND.RATE !
    GET.WINDOW SHOW.CONTROLS ;
: REPITCH. ( -- | resets pitch to normal )
    100 !PITCH.CONTROL
    GET.WINDOW SHOW.CONTROLS ;
: DO.PITCH.PARTS ( -- | handles the different parts of
                  scroll bar )

    LAST.PART
    CASE IN.THUMB      OF TRACK.PITCH.THUMB  ENDOF
        UP.BUTTON      OF -1 DO.PITCH.CONTROL ENDOF
        DOWN.BUTTON    OF 1 DO.PITCH.CONTROL ENDOF
        PAGE.UP        OF -10 DO.PITCH.CONTROL ENDOF
        PAGE.DOWN      OF 10 DO.PITCH.CONTROL ENDOF
    ENDCASE ;
CREATE TIME.MAX 0 ,      ( maximum time recorded on host Mac )
: SET.TIME.MAX ( -- | sets time.max in tenths of a second )
    ?HEAP.SIZE 5000 - 0 MAX 10 * SAMPLE.RATE @ / TIME.MAX ! ;
: @TIME.CONTROL ( -- current value | reads value of time
                  control )

    TIME.CONTROL @ GET.CONTROL ;
: !TIME.CONTROL ( value -- | set time control )
    0 MAX TIME.MAX @ MIN TIME.CONTROL @ SWAP SET.CONTROL ;
: DISP.TIME ( --- | show amt of time to record )
    10 39 MOVE.TO ." Recording Time =" @TIME.CONTROL 10 /MOD
    ." ." . ." secs" 10 spaces ;
: START. ( -- | starts recording )
    START.BUTTON @ IN.BUTTON HILITE.CONTROL
    RSTOP.BUTTON @ 255 HILITE.CONTROL
    RT.MAR @ @TIME.CONTROL SAMPLE.RATE @ * 10 / RECORD.BYTES
    @TIME.CONTROL SAMPLE.RATE @ * 10 / RT.MAR +!

```

```

        START.BUTTON @ 0 HILITE.CONTROL
        RSTOP.BUTTON @ 0 HILITE.CONTROL
        FLUSH.EVENTS
        RECORDING.WINDOW HIDE.WINDOW ;
: RSTOP. ( -- | stops recording )
    RECORDING.WINDOW HIDE.WINDOW ;
: INC.TIME.CONTROL ( increment -- | increments time control )
    @TIME.CONTROL + !TIME.CONTROL ( inc control )
    LAST.PART UPDATE.CONTROL      ( hilight part )
    0      UPDATE.CONTROL      ( unhighlight )
    DISP.TIME      ;
: DO.TIME.CONTROL ( inc -- | handles mouse to increment
                    time control )

    BEGIN MOUSE.BUTTON
    WHILE DUP INC.TIME.CONTROL
    REPEAT DROP ;
: TRACK.TIME.THUMB ( -- | follows thumb on time control )
    FOLLOW.MOUSE DROP
    GET.WINDOW SHOW.CONTROLS
    DISP.TIME      ;
: DO.TIME.PARTS ( -- | handles the different parts of scroll bar )
    LAST.PART
        CASE IN.THUMB      OF TRACK.TIME.THUMB      ENDOF
            UP.BUTTON      OF -1 DO.TIME.CONTROL ENDOF
            DOWN.BUTTON    OF  1 DO.TIME.CONTROL ENDOF
            PAGE.UP        OF -10 DO.TIME.CONTROL ENDOF
            PAGE.DOWN      OF  10 DO.TIME.CONTROL ENDOF
        ENDCASE ;
: RECORDING.SELECT ( -- | checks which selection is pushed )
    LAST.CONTROL DUP TOGGLE.CONTROL
    CASE START.BUTTON @ OF START. ENDOF
        RSTOP.BUTTON @ OF RSTOP. ENDOF
    ENDCASE ;
: DO.RECORDING.MOUSE ( -- | handles mouse driven controls )
    IN.CONTROL?
        IF LAST.CONTROL TIME.CONTROL @ =
            IF DO.TIME.PARTS
            ELSE FOLLOW.MOUSE
            IF RECORDING.SELECT THEN
            THEN
            THEN ;
: RECORDING.PROG ( flag -- | routine for the RECORDING.WINDOW )
    IF SRCCOPY TEXTMODE
    0 !TIME.CONTROL
    SET.TIME.MAX
    TIME.CONTROL @ 0 TIME.MAX @ SET.CONTROL.RANGE
    RECORDING.WINDOW SHOW.WINDOW DISP.TIME
    BEGIN DO.EVENTS MOUSE.DOWN =
        IF DO.RECORDING.MOUSE THEN
        AGAIN
    ELSE RECORDING.WINDOW HIDE.WINDOW
    THEN ;
: SCROLL.GRPH ( -- | scrolls along waveform )
    strt.gph @
    @hpos comp.fact @ * 10 *
    dup strt.gph !
    - comp.fact @ /
    dup abs gph.hsize @ <
    IF gph.box over 0 get.window 122 + @ scroll
        dup 0> IF 0 strt.gph @ rot graph.test
            ELSE dup gph.hsize @ + dup comp.fact @ *
                strt.gph @ + rot negate graph.test
            THEN
        ELSE drop graph.redo

```

```

    THEN ;
: INC.SCROLL.CONTROL ( n -- | increment control by n )
    @HPOS + !HPOS
    LAST.PART UPDATE.CONTROL
    0      UPDATE.CONTROL
    SCROLL.GRPH ;
: DO.SCROLL.CONTROL ( inc -- | handles mouse to increment scroll
                        control )

    BEGIN MOUSE.BUTTON
    WHILE DUP INC.SCROLL.CONTROL
    REPEAT DROP ;
: TRACK.SCROLL.THUMB ( -- | handles the thumb control )
    FOLLOW.MOUSE DROP
    GET.WINDOW SHOW.CONTROLS
    SCROLL.GRPH ;
: DO.SCROLL.PARTS ( part.code -- | performs the part function )
    CASE IN.THUMB OF TRACK.SCROLL.THUMB ENDOF
        UP.BUTTON   OF -1 DO.SCROLL.CONTROL ENDOF
        DOWN.BUTTON OF  1 DO.SCROLL.CONTROL ENDOF
        PAGE.UP     OF -10 DO.SCROLL.CONTROL ENDOF
        PAGE.DOWN   OF  10 DO.SCROLL.CONTROL ENDOF
    ENDCASE ;
: MAIN.SELECTED ( -- | toggle control and do it )
    LAST.CONTROL DUP TOGGLE.CONTROL
    CASE PLAY.BUTTON @ OF HUSH PLAYBACK. ENDOF
        RECORD.BUTTON @ OF HUSH RECORD.  ENDOF
        STOP.BUTTON   @ OF HUSH          ENDOF
        REPITCH.BUTTON @ OF REPITCH.      ENDOF
    ENDCASE ;
: REGION.SELECTION.1 ( -- | selects a data range, and scrolls )
    @mouse.dn pt>offset dup last.mar !
    with.shift? if extend.mar
                  else reset.mar
                  then
    begin still.down
    while @mouse dup
        point>xy drop
        dup      0 > not if -10 inc.scroll.control then
        gph.hsize @ < not if  10 inc.scroll.control then
        pt>offset update.mar
    repeat ;
: DO.MAIN.MOUSE ( -- | handles mouse driven controlsa )
    IN.CONTROL?
    IF IN.SCROLL.BAR?
        IF SWAP DROP DO.SCROLL.PARTS
        ELSE LAST.CONTROL PITCH.CONTROL @ =
            IF DO.PITCH.PARTS
            ELSE FOLLOW.MOUSE IF MAIN.SELECTED THEN
            THEN
        THEN
    ELSE @mouse.dn point>xy swap drop controls.vsize @ >
        IF REGION.SELECTION.1 THEN
        THEN ;
include" fileio blocks"

```

(** NOTE:

fileio blocks is a public domain routine distributed
by NMFUG -- National MacFORTH users group -- and
available through PSG4 on COMPUSERVE

it is a standard Macintosh file IO interface

it contains the following definitions

```
data.open ( opens data file )
data.create ( creates data file )
```

and it creates the following variables

```
Ofile      ( output file )
Ifile      ( input file )
Ifile      ( input file name )
Oname      ( output file name )
```

Parts of the code in small letters are likely to be optimized slightly in release version.

```
** )

: fopen ( -- | opens a data file and retrieves data )
-1 ' ifile ! data.open ifile -1 > if
  rcd.dsize 4 0 ifile read.virtual
  sample.rate 4 4 ifile read.virtual
  repitch.
  0 rcd.dsize @ ?heap.size 5000 - 0 max min
  0 rcd.dsize ! rcd.insert drop
  rcd.handle @@ rcd.esize @ + rcd.dsize @ 8 ifile
    read.virtual
  ifile ' ofile ! ( so that saves will go to this
    file)

  fopening
  iname sys.window set.wtitle TRUE named !
  FALSE changes !
  sys.window show.window sys.window window
  sys.window select.window then ;
: fsaveas ( -- | names a file and saves data to it )
  named @ IF ( one file already open )
  ofile ( save ofile on stack )
  -1 ' ofile ! data.create ofile -1 >
  if ofile rcd.to.file ofile close ofile remove then
  ' ofile ! ( restore ofile )
  ELSE ( no file yet open )
  -1 ' ofile ! data.create ofile -1 >
  if ofile rcd.to.file oname sys.window set.wtitle
  true named ! false changes ! then THEN ;
: fsave ( -- | saves file )
  changes @ IF
  ofile -1 > if ofile delete
    ofile create.file file.error?
    ofile open file.error?
    ofile get.file.info
    "data ofile >fcb +fcb.type !
    "maca ofile >fcb +fcb.appl !
    ofile set.file.info
    ofile rcd.to.file
    false changes !
  else fsaveas
  then THEN ;
: fnew ( -- | opens a new file )
  " untitled " sys.window set.wtitle FALSE named !
  -1 ' ifile ! -1 ' ofile ! TRUE changes !
  fopening
  sys.window show.window sys.window select.window ;
: fclose ( -- | closes a file )
  ( fsave ) ( for now, no saving done here )
```

```

        ofile -1 > if ofile close ofile remove then
        ifile -1 > if ifile close ifile remove then
        0 lt.mar ! 0 rt.mar ! 0 last.mar !
        0 rcd.dsize @ rcd.delete drop
        fclosing
        sys.window hide.window ;
: fquit ( -- | quits program )
    fclose bye ;
: file.menu ( -- | menu creation )
    file.menu.id delete.menu
    0 " File " file.menu.id new.menu
    " New(;Open...;Close;Save;Save As...;-(;Quit"
        file.menu.id append.items
    file.menu.id menu.selection:
        case 1 of fnew endof
            2 of fopen endof
            3 of fclose endof
            4 of fsave endof
            5 of fsaveas endof
            7 of fquit endof
        endcase
    0 hilite.menu graph.redo ;
: MAIN.PROG ( flag -- | main program for main window )
    IF GRAPH.REDO
        BEGIN DO.EVENTS
            CASE IN.CLOSE.BOX OF hide.mark fclose END OF
                IN.SIZE.BOX OF GRAPH.REDO END OF
                MOUSE.DOWN OF hide.mark DO.MAIN.MOUSE
                    show.mark END OF
            ENDCASE
        AGAIN
    ELSE
        THEN ;
: edit..menu ( -- | menu creation )
    edit..menu.id delete.menu
    0 " Edit. " edit..menu.id new.menu
    " Undo(;Cut(;Copy(;Paste(;Clear;-(;Reverse"
        edit..menu.id append.items
    edit..menu.id menu.selection:
        case 5 of delete.selection endof
            7 of flip.selection endof
        endcase
    TRUE changes !
    0 hilite.menu GRAPH.REDO ;
: comp.menu ( -- | menu creation )
    comp.menu.id delete.menu
    0 " Compression " comp.menu.id new.menu
    " x1;x10;x100;x1000" comp.menu.id append.items
    comp.menu.id menu.selection:
        1 FALSE comp.menu.id item.check ( remove check marks )
        2 FALSE comp.menu.id item.check
        3 FALSE comp.menu.id item.check
        4 FALSE comp.menu.id item.check
    dup TRUE comp.menu.id item.check ( add a check mark )
        case 1 of 1 comp.fact ! endof
            2 of 10 comp.fact ! endof
            3 of 100 comp.fact ! endof
            4 of 1000 comp.fact ! endof
        endcase
    0 hilite.menu GRAPH.REDO ;
: MAIN.CONFIG ( wptr -- wptr | reconfigure sys.win )
    >R
    " MacRecorder " I W.TITLE
    CLOSE.BOX SIZE.BOX SCROLL.LEFT/RIGHT + +
    I W.ATTRIBUTES
    40 10 340 500 I W.BOUNDS

```

```

I ON.ACTIVATE MAIN.PROG
R> ;
: APPEND.WINDOWS.ETC ( -- | appends windows, sets up menus
                        and controls )
    APPLE.MENU file.menu edit..menu
    COMP.MENU DRAW.MENU.BAR
    GET.WINDOW PLAY.BUTTON APPEND.CONTROL
    GET.WINDOW RECORD.BUTTON APPEND.CONTROL
    GET.WINDOW STOP.BUTTON APPEND.CONTROL
    GET.WINDOW REPITCH.BUTTON APPEND.CONTROL
    GET.WINDOW PITCH.CONTROL APPEND.CONTROL
    RECORDING.WINDOW ADD.WINDOW
    RECORDING.WINDOW START.BUTTON APPEND.CONTROL
    RECORDING.WINDOW RSTOP.BUTTON APPEND.CONTROL
    RECORDING.WINDOW TIME.CONTROL APPEND.CONTROL
    RECORDING.WINDOW ON.ACTIVATE RECORDING.PROG ;
: MAIN.SETUP ( -- | level 3 MacFORTH standalone turnkey
              token )
    GINIT
    0 CONTROLS.VSIZE @ XYOFFSET
    APPEND.WINDOWS.ETC
    INIT.ALL ;

(*** PROGRAM ADC1:: *** )

FORTH DEFINITIONS
20000 MINIMUM.OBJECT ( establish minimum object & Vocab )
5048 MINIMUM.VOCAB
( INCLUDE" Controls" )
( INCLUDE" Assembler" )
( "controls" and "assembler" blocks files are standard
  with Level 2 MacFORTH )
create snd.rate 0 , ( sound rate for playback)
create sample.rate 0 , ( rate of the digitizer )
CREATE flip.table 256 allot ( a lookup table )
create comp.fact 10 , ( compression factor for graphics )
create rcd.handle 0 , ( handle to sound record data )
create rcd.dsize 0 , ( sound record data size )
create rcd.esize 6 , ( extra size -- six bytes -- for
                     std sound record )
create lt.mar 0 , ( left margin in sound record
                  data )
create rt.mar 0 , ( right margin in sound record
                  data )
create last.mar 0 , ( last margin selected in sound
                    record dta )
create gph.hsize 500 , ( horizontal size of graphics
                       region )
create gph.vsize 280 , ( vertical size of graphics
                       region )
create strt.gph 0 , ( offset for start of graph )
0 0 0 0 rect gph.box ( size of graphics box )
create controls.vsize 0 , ( vertical size of controls )
11 constant file.menu.id
12 constant edit..menu.id
13 constant comp.menu.id
create changes FALSE , ( boolean for whether changes
                        made )
create named FALSE , ( boolean for whether file has
                      been named )
create init.length 0 , ( initial length of file )
hex ( various system constants for the SCC chip )
9FFFF8 constant SCCRBase ( SCC base read address )
BFFFF9 constant SCCWBase ( SCC base write address )

```

```

        6 constant aData      ( offset for A channel data      )
        2 constant aCtl      ( offset for A channel control    )
decimal
: sccpeek ( n -- read[n] | to read value in read register n )
    sccwbase aCtl + c! sccrbase aCtl + c@ ;
: sccpoke ( n1\n2 -- | to write value n1 into write reg. n2 )
    sccwbase aCtl + c! sccwbase aCtl + c! ;
: SCC.INIT ( -- | initializes the SCC chip )
136 15 sccpoke ( turn off the CTS interrupt )
    3 1 sccpoke ( turn off the Rx interrupt )
    0 14 sccpoke ( null command )
    48 11 sccpoke ( use TRxC as the receiver clock only )
    0 10 sccpoke ( null command )
    8 9 sccpoke ( initialize master reset )
    98 5 sccpoke ( initialize transmitter )
    68 4 sccpoke ( x16 clock mode, trans. 1 stop bit, no par )
193 3 sccpoke ( initialize receiver, 8 bits ) ;
( creating a purgeable assembler )
CREATE OTHER.DP 0 , ( creates alternate dictionary pointer )
: SWAP.DP ( -- | swapping dictionary pointers, see Level 2 doc )
    DP @ OTHER.DP @ DP ! OTHER.DP ! ;
HERE 4096 + OTHER.DP ! SWAP.DP
: ?ABOVE ( -- | see Level 2 doc )
    LATEST 2-
    BEGIN 2DUP >W@< DUP
    WHILE TOKEN>ADDR <
        IF >W@< EXIT THEN
            2+ COUNT 31 AND +
        REPEAT 2DROP NOT ;
: PRUNE ( -- | removes excess definitions )
    BEGIN ?ABOVE DUP
    WHILE DUP DEALLOT BEHEAD
        REPEAT 2DROP ;
INCLUDE" Assembler"
SWAP.DP ( swapping dictionary pointers )

CODE MAKE.FLIP.TABLE ( addr -- | makes the lookup table at addr )
    A0 POP, ( addr )
    256 # A0 WORD ADDA, ( starts at top of the table )
    255 # D0 WORD MOVE, ( counter for outer loop )
    BEGIN,
        D0 D1 BYTE MOVE, ( d1 gets untransformed value )
        07 # D3 WORD MOVE, ( counter for inner loop )
        BEGIN,
            D1 01 # BYTE LSR, D2 01 # BYTE ROXL,
                ( d2 gets transformed value )
            D3 WLOOP,
            D2 A0 -) BYTE MOVE, ( store value in table )
        D0 WLOOP, NEXT END-CODE

CODE RECORD.ASS ( addr\cnt\addr1 -- | load cnt bytes from the
    serial port to addr; addr1= lookup table )
    A1 POP, D0 POP, A0 POP, ( table addr, count, addr )
    D1 LONG CLR,
    D0 01 LONG SUBQ, ( pre-decrement count )
    BEGIN, ( begin the input loop )
    BEGIN, ( begin the wait for Rx char loop )
        SccRBase aCtl + @#L 00 # BTST, NE ( check Scc for Rx char )
    UNTIL,
        SccRBase aData + @#L D1 BYTE MOVE, ( RX char goes to D1 )
    D1 00 WORD A1 @I) A0 )+ BYTE MOVE, ( translate via lookup tab )
    D0 LOOP,
    NEXT END-CODE

```

```

CODE GRAPH.COMP ( scrn\addr\cnt\fact -- | graphs min to max )
D0 POP, D6 POP, A0 POP, D4 POP, D0 PUSH, ( save fact on stack )
BEGIN,
    SP ( )          D0 LONG MOVE, ( get fact on stack )
    D0              01 LONG SUBQ, ( predecrement counter )
    255 #           D1 LONG MOVE, ( initial min )
    0 #             D2 LONG MOVE, ( initial max )
    BEGIN,         D3 LONG CLR, ( clear upper bytes )
                  A0 )+ D3 BYTE MOVE, ( get value at addr )
                  D1 D3 WORD CMP, ( compare for a min )
    MI IF, D3       D1 WORD MOVE, THEN, ( save min )
                  D2 D3 WORD CMP, ( compare for a max )
    PL IF, D3       D2 WORD MOVE, THEN, ( save max )
    D0 LOOP,        ( do min\max loop )
    A0 D5 LONG MOVE, ( Preserve A0 in D5 )
    D4 PUSH, D2 PUSH, D4 PUSH, D1 PUSH, ( X\Ymax\X\Ymin )
    IP RP -) LONG MOVE, ( protect IP )
                  ( D4 D5 D6 automatically protected )
    >FORTH VECTOR >CODE ( graph min to max )
    RP )+ IP LONG MOVE, ( restore IP )
    D5 A0 LONG MOVEA, ( restore A0 )
    D4 01 LONG ADDQ, ( increment scrn )
    D6 01 LONG SUBQ, EQ ( decrement count )
UNTIL,
    D0 POP, ( discard fact, which was still on stack )
NEXT END-CODE

CODE FLIP.ASS ( addr1\addr2 -- | reverse order of data bytes )
A2 POP, A1 POP,
BEGIN, A1 ( ) D0 BYTE MOVE,
      A2 ( ) A1 )+ BYTE MOVE, ( A1 is incremented )
      D0 A2 ( ) BYTE MOVE,
      A2 1 LONG SUBQ, ( A2 is decremented )
      A1 A2 LONG CMPA, ( A2 - A1 )
MI UNTIL, ( stop if A2 < A1 )
NEXT END-CODE
." Purging assembler"
CREATE MARKER ' ASSEMBLER @ PURGABLE ' FLIP.ASS PRUNE ( purges
                                                         assembler )
: graph1 ( scrn\addr\cnt -- | graphs cnt bytes, starting at addr )
  3 pick 3 pick c@ move.to ( position pen on screen )
  over + swap do ( loop from addr to addr+cnt )
  dup i c@ draw.to 1+ loop ( draw, and increment scrn )
  drop ; ( clear stack )
: rcd.delete ( d\cnt -- Bool | will delete cnt bytes from the data
               rcd starting at offset d.
               Space returned to the heap. )
  over rcd.handle @@ rcd.esize @ + + ( offset d )
  2dup + swap ( offset d + cnt )
  rcd.dsize @ 5 roll - 4 pick - cmove ( delete cnt )
  negate rcd.dsize +! ( decrement dsize by cnt )
  rcd.handle @ rcd.dsize @ rcd.esize @ + resize.handle
  ( return extra room to the heap )
  if FALSE else TRUE then ;
  ( TRUE if successfully returned to heap )
: rcd.insert ( d\cnt -- Bool | will insert cnt bytes into the data
               rcd starting at offset d.
               Returns TRUE if successful )
  rcd.handle @ over rcd.esize @ rcd.dsize @ + +
  resize.handle if ( increase space to handle by cnt )
  2drop FALSE ( unsuccessful )
  else
  over rcd.handle @@ rcd.esize @ + + ( offset d )
  2dup + ( offset d + cnt )

```

```

        rcd.dsize @ 5 roll - cmove>          ( make space )
        rcd.dsize +!                          ( increment dsize by cnt )
        TRUE                                  ( successful )
            then ;
hex
A403 OS.TRAP A.WRITE ( buf.ptr -- | executes device write )
decimal
: long.aplay ( lnth\addr -- | pass the addr and lnth of a sound
                        synth. record to the sound driver
                        and plays it asynchronously )
    sound.fcb          ( a file control block in FORTH )
    0      over 12 + !   ( ioCompletion ptr )
    65532 over 24 + w!   ( ioRefNum = -4 )
    swap   over 32 + !   ( ioBuffer ptr = addr )
    swap   over 36 + !   ( ioRegCount = lnth )
    0      over 44 + w!   ( ioPosMode )
    0      over 46 + !   ( ioPosOffset )
    a.write             ( execute the device write call ) ;
: record.bytes ( d\cnt -- | record cnt bytes at offset d in data
                    rcd)
    dup 0= if 2drop          ( if cnt=0, done )
    else 2dup rcd.insert     ( insert space )
        if swap rcd.handle @@ rcd.esize @ + + ( offset d )
            swap flip.table record.ass      ( input cnt bytes)
        else 10 39 move.to ." insufficient heap space "
        then then ;
: play.bytes ( d\cnt -- | plays cnt bytes from data rcd, at
                    offset d)
    swap dup 2 mod - swap          ( make d even )
    over rcd.handle @@ + pad rcd.esize @ cmove ( save 6 bytes )
    0      rcd.handle @@ 4 pick + w!   ( freeform mode )
    snd.rate @ rcd.handle @@ 4 pick + 2+ ! ( sndrate longword)
    rcd.handle @ lock.handle
    6+ rcd.handle @@ 3 pick + long.aplay ( play the record )
    rcd.handle @ unlock.handle
    rcd.handle @@ + pad swap rcd.esize @ cmove ;
: graph.test ( scrn\offset\cnt -- | graph starting at scrn from
                    offset in rcd. Goes up
                    to cnt, or end of rcd. )
    rcd.dsize @ 3 pick - comp.fact @ /
    min dup 0>                ( compare cnt to end of rcd )
    IF 3 pick 3 pick 3 pick    ( scrn\off\cnt\s\o\c )
        swap rcd.handle @@ rcd.esize @ + + swap ( s\o\c\s\addr\c)
        comp.fact @ 1 = IF graph1
            ELSE comp.fact @ graph.comp
            THEN
                ( scrn\off\cnt )
            rt.mar @ 3 pick - comp.fact @ / min 3 pick +
            lt.mar @ rot - comp.fact @ / 0 max rot + swap
            2dup < IF 0 swap 260 invert rectangle
            ELSE 2drop THEN
    ELSE 2drop drop THEN ;
: graph.clear ( -- | clears out the graphing area )
    gph.box erase.rect ;
: round ( n -- n' | round down to a multiple of comp.fact )
    comp.fact @ / comp.fact @ * ;
: to.hpos ( n -- n' | converts an offset to a hbar value )
    comp.fact @ / 10 / ;
: pt>offset ( pt -- n | translate point on screen to offset)
    point>xy drop 0 max gph.hsize @ min ( keep x value on screen)
    comp.fact @ * strt.gph @ + 0 max rcd.dsize @ min ;
: offset>scrn ( n -- n' | translate an offset to scrn point )
    strt.gph @ - comp.fact @ / 0 max gph.hsize @ min ;
: with.shift? ( -- Bool | returns true if shifted mouse down )

```

```

        mouse.down.record 16 + w@ 512 and if TRUE else FALSE then ;
: offset.invert ( offset1\offset2 -- | inverts on graph )
    offset>scrn 0 rot offset>scrn 280 invert rectangle ;
variable mark.state mark.state off
: invert.mark ( -- | change the mark on screen at lt.mar )
lt.mar @ dup comp.fact @ + offset.invert
    -1 mark.state @ - mark.state ! ;
: hide.mark ( -- | remove mark line at lt.mar )
mark.state @ if invert.mark then ;
: show.mark ( -- | place a marker line at lt.mar )
lt.mar @ rt.mar @ = mark.state @ not AND if invert.mark then ;
: graph.redo ( -- | redoes the entire graph )
    hide.mark      ( finds the current graph area size )
    GET.WINDOW +WBOUNDS DUP 4+ W@      ( vertical window size )
    CONTROLS.VSIZE @ 0 MAX - gph.vsize ! ( vert gph region size )
    6+ W@ gph.hsize !      ( horizontal window size )
    CONTROLS.VSIZE @ 0
    gph.vsize @ 3 PICK + gph.hsize @ gph.box !rect ( gph region )
    100 gph.vsize @ 100 * 260 / xyscale ( scale to gph region )
    0 -1 gph.hsize @ -1 vector ( line below control region )
    strt.gph @ rcd.dsize @ >
    IF lt.mar @ strt.gph ! THEN ( enforce strt <= dsize )
    strt.gph @ round strt.gph !      ( round down )
    GET.WINDOW +HBAR @ 0 rcd.dsize @ to.hpos SET.CONTROL.RANGE
    strt.gph @ to.hpos !hpos ( update the scroll thumb )
    GET.WINDOW SHOW.CONTROLS ( redraw all controls )
    graph.clear 0 strt.gph @ gph.hsize @ graph.test show.mark ;
: reset.mar ( offset -- | resets lt.mar and rt.mar )
    lt.mar @ rt.mar @ offset.invert ( uninvert last selection )
    dup lt.mar ! rt.mar ! ;
: extend.mar ( offset -- | extends the appropriate margin )
    dup lt.mar @ < IF lt.mar @ over lt.mar !
        ELSE rt.mar @ over rt.mar ! THEN
        offset.invert ;
: update.mar ( offset -- | updates the margins )
    dup last.mar @ offset.invert ( invert from last offset )
    dup rt.mar @ last.mar @ = if rt.mar ! else lt.mar ! then
    last.mar !
    rt.mar @ lt.mar @ < if rt.mar @ lt.mar @ rt.mar ! lt.mar ! then
    ;
: FLIP.SELECTION ( -- | reverse data between lt.mar and rt.mar )
    LT.MAR @ RT.MAR @ <
    IF RCD.HANDLE @@ RCD.ESIZE @ + DUP
        LT.MAR @ + SWAP RT.MAR @ + FLIP.ASS
    THEN ;
: DELETE.SELECTION ( -- | delete data between lt.mar and rt.mar )
    LT.MAR @ RT.MAR @ <
    IF LT.MAR @ RT.MAR @ OVER - RCD.DELETE drop
        lt.mar @ rt.mar !
    THEN ;
: INIT.ALL ( -- | initializes SCC chip and various constants )
    SCC.INIT
    9600 sample.rate ! ( initialize input rate )
    sample.rate @ 65536 * 22257 / snd.rate ! ( playback rate )
    flip.table make.flip.table
    rcd.esize @ from.heap rcd.handle ! ( initialize space for
        rcd ) ;
: RCD.TO.FILE ( file# -- | saves the record in file # )
    rcd.dsize      4      0 4 pick write.virtual
    sample.rate    4      4 4 pick write.virtual
    rcd.handle @@ rcd.esize @ +
        rcd.dsize @ 8 4 pick write.virtual
    drop
    ( should make a check here to see it recorded ) ;

```