Recovering Data From 40 Year Old Cassette Tapes

My first micro-computer was an Explorer 85 kit from Netronics Research, which used standard audio cassette tapes for storage. Other storage methods were available at the time, but were far more expensive than tape. Even floppy drives were beyond my budget back then. Eventually prices came down and I bought a 64K CP/M system with dual 5 1/4 inch floppy drives. The Explorer 85 and some of the cassette tapes got moved from closet to closet and box to box over the years, but never thrown out. I was cleaning out the last closet they were in and thought it would be interesting to see if anything could be recovered from the cassette tapes.

My Explorer 85
S100 Boards, Top Down:
Bus Monitor+Printer Port,
Magnetic Bubble Memory
32K Static RAM

Although the old Explorer 85 still worked, the cassette recorder didn't. I do have a newer one that works, but there are two major problems with using it to read files with the Explorer 85. The biggest is that the new recorder doesn't have a turns counter like the old one had. All the files on the cassette tapes were indexed and located by the counter. The other problem is that files got recorded with a single character or byte ID, which needed to be supplied when reading that file. If the ID wasn't correct the Explorer 85 would simply report an error just like any other tape error. I always wrote the start and end counter values for each file on each cassette, but I'm not sure about the ID. I was able to successfully save and load data using the new recorder and an old, unused cassette, so I knew all the hardware was working.

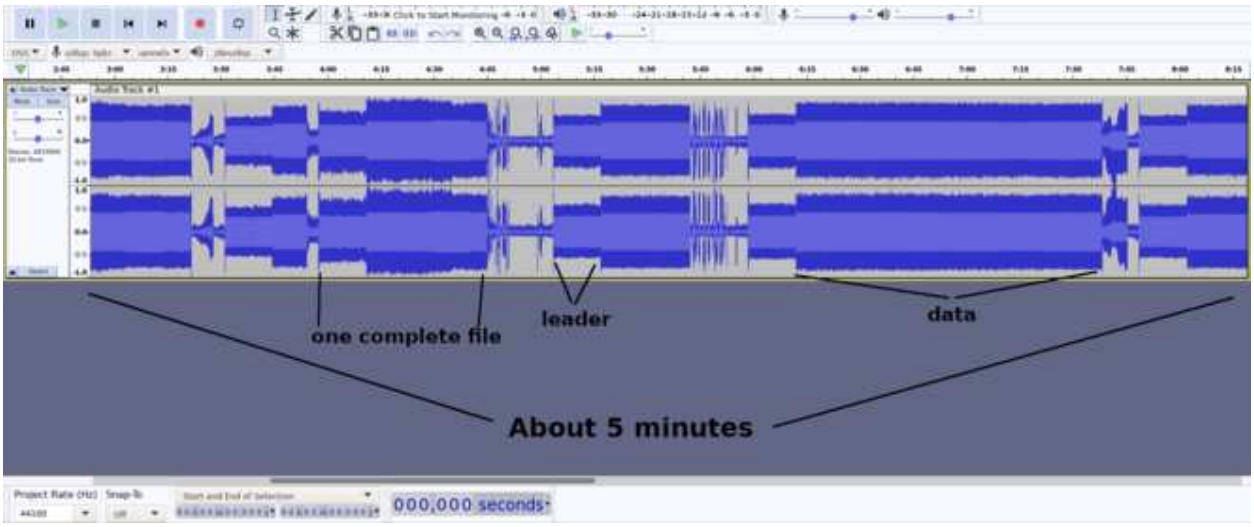
Cassette Tape Label
Showing Index, Name, Address (and ID?)

Uncertainty about positioning and ID for each file made it unlikely that I could recover anything using the computer itself. I was also reluctant to put 40 year old cassettes under any more stress than absolutely necessary. So I had to fall back to plan B, which was to digitize the tapes and see if any sense could be made from the digital images. Using an inexpensive analog audio to USB converter, I captured one complete side of one tape. The software I used was Audacity 3 running on a FreeBSD 12.3 system. I used default settings for everything.

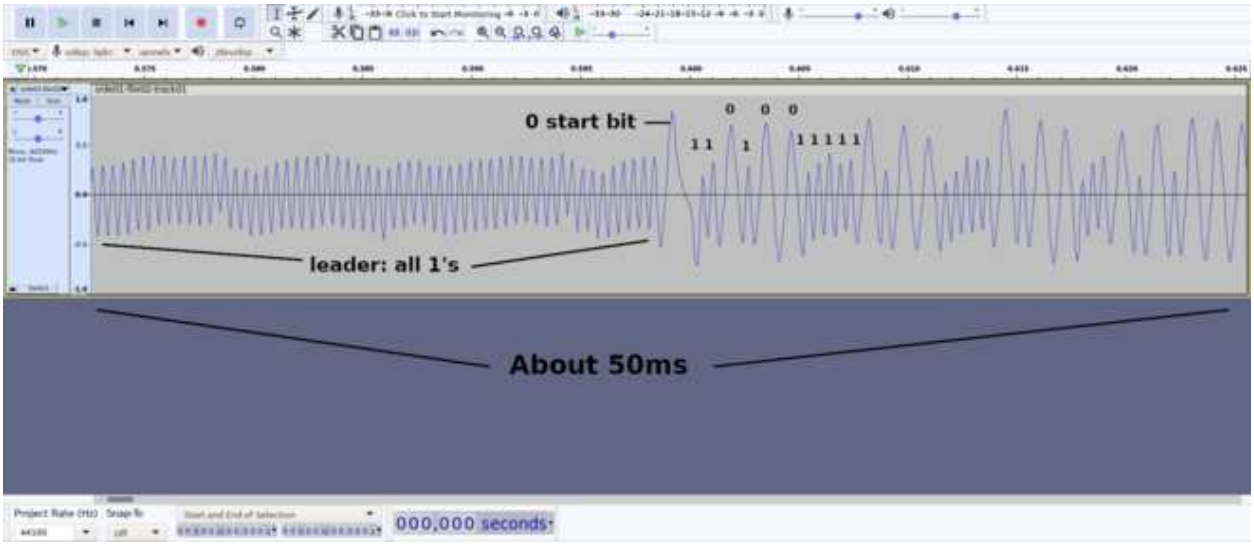
At this point I needed to know how Netronics recorded data on tape. There were a lot of different formats in use back then, many of which are well documented, but I couldn't find anything specific to the Explorer 85. However, Netronics also sold an Elf II product with a cassette interface, which is fairly well documented. It seemed reasonable to hope that they would have used a similar scheme for both products, especially since the hardware involved looked similar for both.

The Elf II used a leader of 1's, followed by a single 0, followed by two bytes of address, followed by two bytes of size. All bytes were recorded most significant bit first, using even parity. Each bit was represented as a single pulse with 0's being significantly longer than 1's. The exact timing doesn't matter here.

Looking at the captured audio with Audacity, I could clearly see an initial uniform leader, followed by a single longer pulse. When I assumed that short pulses were 1's and long ones were 0's, and each byte was nine pulses with even parity, I could easily figure out the first few bytes. It turned out that the Explorer 85 tape format was very much like the Elf II, except that the Explorer 85 included an ID byte, and used start and end address instead of start address and size.



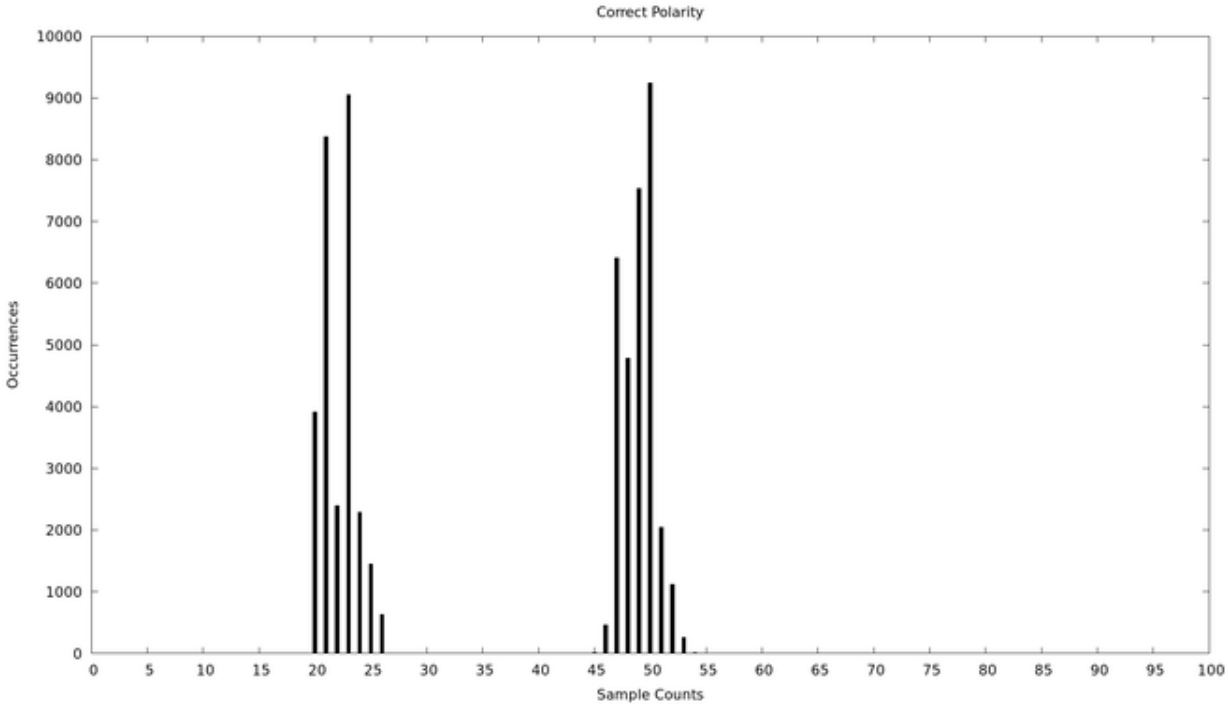
Showing Some Of The Files On One Tape



Showing Leader of 1's, With Single 0 Start Bit
Followed By 110100111 (D3), etc.

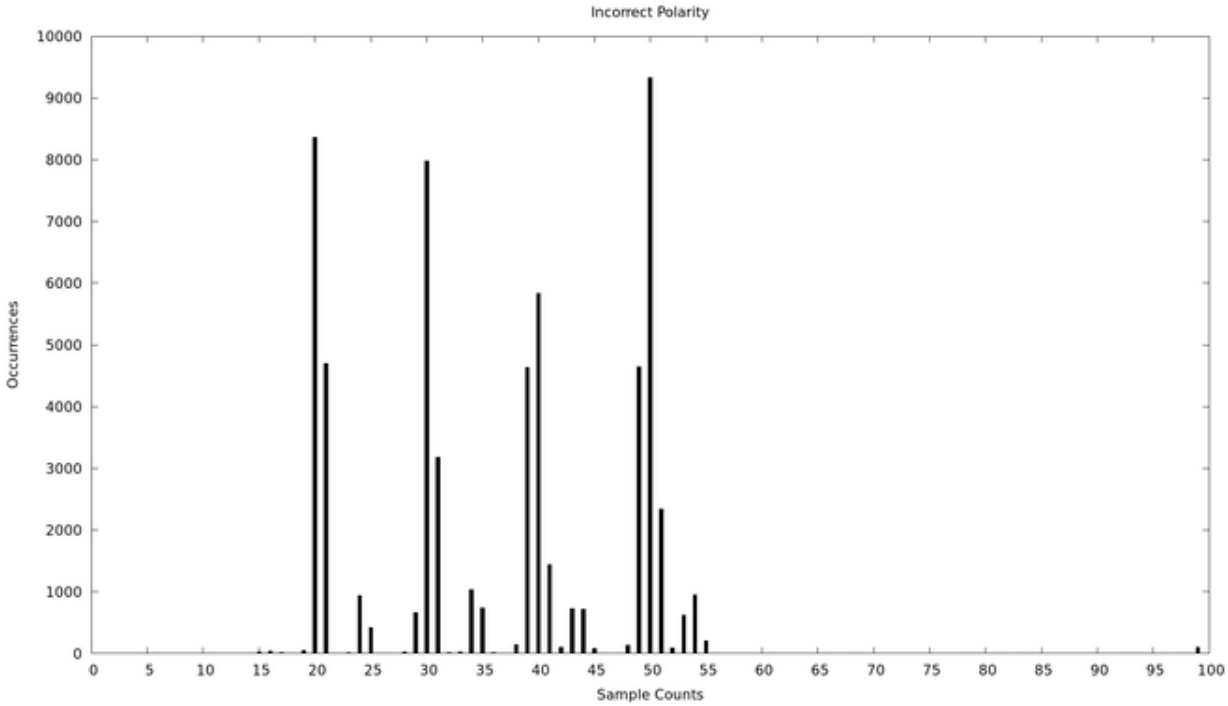
It now looked like it might actually be possible to recover something useful from those 40 year old tapes, but I needed to understand the WAV file structure first. It turns out that WAV files are pretty simple: there's a 44 byte header containing a variety of information, followed by the actual audio data in the form of samples taken at regular intervals. Since I used signed 16 bit samples, I would need to read them back in that form. I wrote a short test program to extract the original tape header information: ID, start address, end address, and tried it out on a number of files. They all reported results that matched my original records, which was very encouraging, so I went ahead and split all the identifiable tape files into separate monaural WAV files.

The simplest approach to recover the original data from the WAV files was to look for a change in sign from negative to positive and count the number of samples between each. Smaller counts should be 1's and larger counts should be 0's. That worked with only limited success. Most files reported hundreds of parity errors, and only the shortest files converted without error. When I printed out a histogram of counts and occurrences I could see peaks around 20 and 50. For some tape files there were also counts well below 20 and well above 50, so I added low and high cut-offs. That almost magically reduced the number of errors to zero for many of the cassette files.



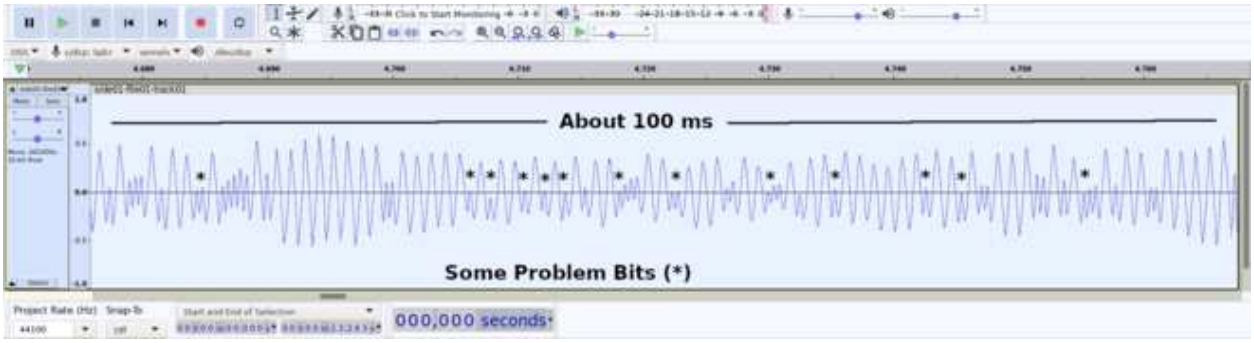
Histogram With Correct Polarity
With Two Well Defined Peaks

However, some of the files I was most interested in were still converting with far too many parity errors. When I looked at the histogram of sample counts I would normally see peaks in the low 20s and 50s, but if the polarity was wrong there were multiple peaks. This was helpful in eliminating that particular variable. When I carefully examined the sample counts for each bit of the bytes surrounding initial parity errors I found something interesting. In each case I examined, it looked like there was an extra 1 bit that probably should have been combined with one of the bits on either side. In other words, there were two 1's with low sample counts, or a 0 and a 1 with low counts. Since I was looking at a text file in each case, I could tell what the byte in question should have been. Unfortunately, there's no consistent way to tell which bit might be causing a problem. It could even be one in a previous byte, so I resorted to arbitrarily dropping a bit and inserting a marker in the output. When it's a text file being processed it's not too hard to manually figure out what that letter should be. That isn't true for binary files of course, but all the files I was most interested in were text.



Histogram With Incorrect Polarity
Showing Multiple Peaks

In the end, adding a low count cutoff, a high count cutoff, and skipping a bit for any remaining parity errors, resulted in an almost 100% recovery for almost all files. And the few parity errors that did occur did not cause a cascade of additional parity errors. There were a small handful of cassette files that still converted with a few errors, but it was pretty easy to manually correct those. And one or two files had so much distortion in a few places that a fairly significant chunk of the file was unrecoverable. Nothing I tried fixed that, nor could I find a way to fix one cassette that had far too many "bad" spots. In fact, most of my efforts with Audacity's signal processing tools just made things worse. There are probably things that would work, but I was already happy with the results at that point and was ready to move on to another project.



Examples Of Some Probable Bit Detection Problems