Digital Archeology: Recovering Digital Objects from Audio Waveforms

Mark Guttenbrunner, Mihai Ghete, Annu John, Chrisanth Lederer, Andreas Rauber
{guttenbrunner, rauber}@ifs.tuwien.ac.at
Vienna University of Technology, Vienna, Austria
http://www.ifs.tuwien.ac.at/dp

Abstract

Specimens of early computer systems stop working every day. It is necessary to prepare ourselves for the upcoming situation of having storage media and no working systems to read data from these carriers. With storage media residing in archives for already obsolete systems it is necessary to extract the data from these media before it can be migrated for long term preservation.
One storage medium that was popular for home computers in the 1980s was the audio tape. The first home computer systems allowed the use of standard cassette players to record and replay data. Audio tapes are more durable than old home computers when properly stored. Devices playing this medium (i.e. tape recorders) can be found in working condition or can be repaired as they are made out of standard components. By re-engineering the format of the waveform the data on such media can then be extracted from a digitized audio stream.
This work presents a case study of extracting data created on an early home computer system, the Philips G7400. The original data formats were re-engineered and an application was written to support the migration of data stored on tapes without using the original system. This eliminates the necessity of keeping an obsolete system alive for preserving access to data on storage media meant for this system. Two different methods to interpret the data and eliminate possible errors in the tape were implemented and evaluated on original tapes recorded 20 years ago. Results show that with some error correction methods parts of the tapes are still readable, even without the original system. It also becomes clear, that it is easier to build solutions now when the original systems are still available.

Introduction

With storage media technology constantly changing, devices for reading certain media become obsolete. However, not only the devices, but also specimens of the computer systems needed to access the data stop working every day. Accessing data on an 8-inch floppy disc requires not only a disc drive, but also a system to connect it to, together with the right software to interpret the data on a logical level. This all is necessary to access the information contained on the floppy disc.

Estates submitted to archives now and in the future not only contain analogue data on paper but also include digital data (e.g. digital diaries) on various storage media. As we have to deal with this data at the time it is ingested in the archive, we have to find methods to extract data from already obsolete storage media and make sure we are able to migrate it to a format that can be used in the digital archive.

A popular storage media for home computers in the early 1980s was the audio tape, also called a compact cassette[1]. As devices that could read and write audio tapes were readily available in most households for recording and playing music, some home computer systems featured connectors for the headphone and microphone jacks of such audio systems that could be used to store and retrieve data. The data was first converted into an analogue waveform by the computer system and then recorded to a standard magnetic audio tape through the microphone input of the audio device. To retrieve the data, the tape was played back on the audio device and the audio waveform was read by the computer system through the headphone connector. The waveform then had to be digitized and the data decoded into a binary stream that was then usable by the original system.

As there was no standardized format across the different home computer platforms, tapes are usually only readable by the system that was used to write them. To retrieve the data from such storage media today we therefore need access to the original system, a tape drive and a person with the knowledge of handling the original system. As audio tape playing devices are still readily available and made of standard components, it is fairly easy to track down a device able to replay tapes. Getting a working specimen of the original system as well as a person to handle the system can be a more difficult task.

In this paper we present a methodology and an application we developed that allows us to retrieve data written by the Philips G7400 home computer system without access to the original system. By using a tape drive and a common personal computer with a sound card we record the audio output of the tape drive and decode the waveform. The resulting bit-stream is then interpreted and the data is migrated to a non-obsolete format that can be ingested into an archival system. Our application may also be used for carrier refreshment - storing the original data again onto an audio tape, correcting errors introduced by the decay of the waveform on the original tape.

[1] Compact Cassette – Wikipedia
http://en.wikipedia.org/wiki/Compact_Cassette

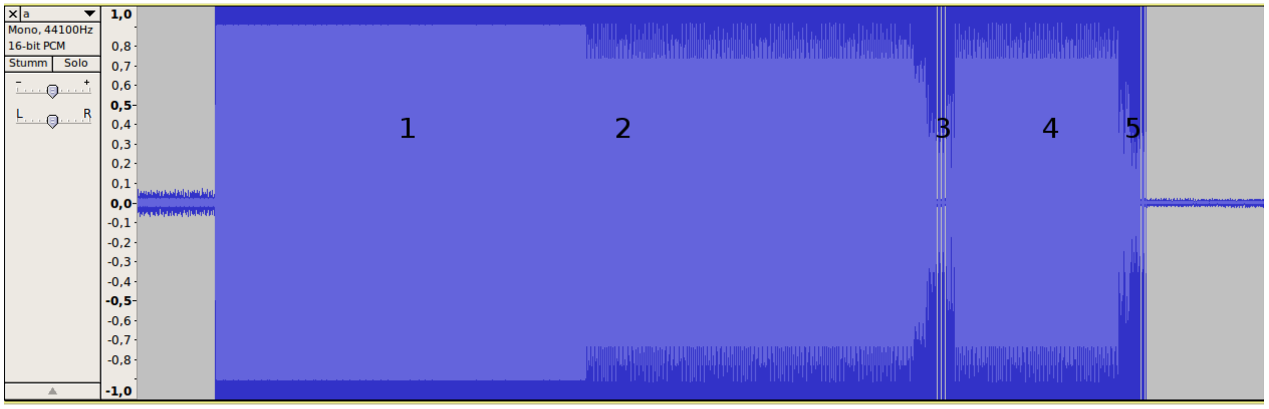


Figure 1: Waveform of sample program (1: 6 kHz lead-in; 2: 256 x 0xFF; 3: file header; 4: 128 x 0xFF; 5: data)

We first present related work on the topic. In the following section we explain how the physical format was reengineered. After that we show how the different types of data are stored logically. Next we present an application we developed that allows us to migrate data from waveforms, together with the two methods we used to convert the recorded waveform into binary data. We show the results of using each method on original tapes, including how much of the data we were able to recover. Finally we present our conclusions.

Related work

The UNESCO guidelines for the preservation of digital heritage (Webb 2005) list four layers where digital data can be threatened.

Audio tapes are magnetic tapes and are subject to various threats on the **physical** level as described in (Bhushan 1992). By converting the analog waveforms to digital waveforms and storing them as digital audio-files on current systems we can avert the immediate threat on the physical layer.

To prevent loss of data on a **logical** level it is necessary to re-engineer the encoding of digital bits in the analog audio signal. In (Ross 1999) an experiment with a Sinclair Spectrum is described, where audio data was migrated to a corresponding binary stream, which could then be interpreted using an emulator of the real system.

However, to separate the digital objects from their original environment the bit-streams have to be interpreted in such a way as to extract the **conceptual** object from the logical bit-stream. By extracting the content and saving it to a format which is not obsolete at the time of migration we can transform the data to a format that is accessible without the original hardware. No expert is needed to operate the original system as it is necessary with emulation as a preservation strategy.

The **essential elements** of the digital object can then be added on ingest in an archival system.

On the system we used for our case study BASIC was used as a main programming language. Source code is a significant property of software and can be necessary to interpret the data stored by applications and is also necessary if software is migrated for preservation purposes (Matthews et. al. 2008). As the G7400 is used as a video game console as well, some of the programs are video games. This provides us with a situation where migration would be a possible solution to preserve some video games for the system (Guttenbrunner et. al. 2008).

Re-engineering the Waveform

For our case study we decided to use the Philips G7400[2]. Originally designed as a video game system with a keyboard, it can be extended to become a home computer with a Microsoft BASIC operating system using the C7420 expansion cartridge. This cartridge also features 3 connector cables for data input, data output and a remote controlling signal used to start and stop the audio tape, if supported by the tape player. The system was chosen as it is already very hard to find specimens in working condition, so there is an imminent threat of losing the data saved with this system permanently. It also met our second criterion: off-the-shelf audio recorders and tapes could be used for storage purposes.

Data on the system can be stored in various formats. The BASIC programming language variant that comes with the system supports saving program listings, screenshots, and storing and retrieving self-defined data (text strings and number arrays) using various forms of the "CSAVE" instruction.

In order to start re-engineering the storage encoding, the original machine's output was connected to the input of a PC's soundcard. We started by writing some test programs on the original machine and recording the resulting audio files using Audacity[3]. One resulting waveform can be seen in Figure 1. By recording different test programs we were able to find out that there is always a 2.77 second lead-in frequency of a 6 kHz sine wave. The data block is stored in a 4.8 kHz sine wave encoding bit set ('1') as a tone and bit cleared ('0') as

[2] Philips G7400- Wikipedia http://en.wikipedia.org/wiki/Philips_Videopac_G7400
[3] Audacity: http://audacity.sourceforge.net/

silence. Every byte is encoded as one start bit (tone), followed 8 data bits (storing least significant bits first) and 2.5 stop bits (silence) (Figure 2). The data is stored at a rate of 1200 bits per second. Every file consists of the following data-bytes in the following order:

No. of Bytes	Code	Contained Information
256	0xFF	<start of file>-signature
32		file-header
128	0xFF	separate header / data
<variable size>		data-block
10	0x00	<end of file>-signature

During our online-research we also found an active community[4] that is still using and also programming this system. One of its members, René van den Enden[5], had written small programs that allowed BASIC programs to be transferred between the original system and a PC. On request he provided us a copy of the source code of his programs which confirmed part of our research regarding the format and provided more details we had not figured out at this point of our investigation.

Re-engineering File Formats

The C7420 is able to store 5 different kinds of data with the following commands:

Object type	Command
BASIC Programs	CSAVE
Screenshots	CSAVES
Arrays	CSAVE*
Strings	CSAVEX
Memory Dumps	CSAVEM

By saving different kinds of test data we were able to identify the format of the 32 byte file header:

- 10 bytes 0xD3
- 1 byte determining the format of the file, usually the character after "CSAVE" (e.g. 'S' for screenshot, 0x20 (Space) for BASIC program)
- 6 bytes for the program name
- 1 byte 0x00
- 5 bytes ASCII characters of the line number at which the execution of the program should start
- 3 bytes 0x00
- 2 bytes start address in memory (Least Significant Byte (LSB) first)
- 2 bytes length of data block in bytes (excluding the first leading byte 0x00, LSB first)
- 2 bytes checksum: all data bytes added up to a 16 bit value (LSB first)

The data block starts with 0x00 and continues depending on the file format with the following data.

[4] Videopac / Odyssey2 Forum: http://videopac.nl/forum/
[5] Rene's VIDEOPAC page: http://home.hetnet.nl/~rene_g7400/

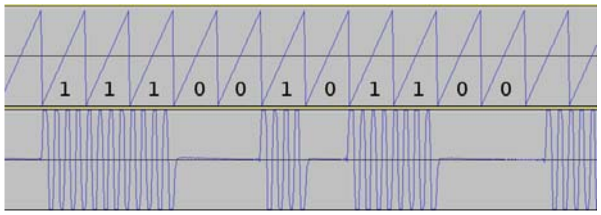


Figure 2: Representation of one byte in the waveform (1 start bit (1), 8 data bits (least significant first: 11010011b = D3h), 2.5 stop bits (0))

Basic Program
For BASIC programs, the data block is split up into lines which contain the following information:

- 2 bytes RAM address of the next BASIC line (LSB first)
- 2 bytes line number (LSB first)
- The actual line with the BASIC commands represented by byte codes between 0x80 and 0xDF
- 1 byte 0x00

At the end of the BASIC program data block 2 bytes 0x00 are written.

Example BASIC line and encoding:
`10 PRINT "HELLO"`

`CF 88` = 0x88CF (address of next BASIC line in RAM)
`0A 00` = 10 (line number)
`94` = PRINT
`20` = <SPACE>
`22 48 45 4C 4C 4F 22`= "HELLO"
`00` = End of line

Screenshot
The C7420 has a text and a graphic mode to display formatted data on screen. For each of the 40x23 positions on the screen the following byte codes are stored in memory and also in the saved data:

- one character-byte
  < 0x80: ASCII character in text mode or special graphical character in graphics mode (as defined in the C7420 user manual)
  >= 0x80: special control or custom character (as defined in the C7420 user manual)
  0x80 changes the background color to the color defined in the foreground bits in the format byte in text mode
- one format byte which changes the attributes of the character or, in the case of 0x80, all following characters:

7	6	5	4	3	0-2
text mode (0)	negative	dbl. width	dbl. height	!blink	foreground color
graph. mode (1)	background color			!blink	foreground color

Array
The first byte of an array encodes the number of dimensions of the array. For each dimension, two

subsequent bytes encode the number of fields in the dimension (LSB first). Finally, for every entry in the array, 4 bytes are used to express the value in different formats, depending on whether the array contains strings or numbers.

String Array:

- 1 byte length of the string in bytes
- 1 unused byte
- 2 bytes address of the string in memory

Note that the actual string data is not saved in the array but the strings have to be saved and loaded separately using the string save command "CSAVEX".

Number Array:
By saving number arrays on the original system, examining the resulting byte stream, changing values and re-loading the array onto the original system we were able to find out that a floating point format is used to store numbers. The encoding is similar to, but does not follow the IEEE 754 floating point standard (IEEE 1987), as that was released 2 years later than the C7420 cartridge. With further testing, the bits for mantissa, sign and exponent were determined. 4 bytes are used to encode the number as a 32 bit floating point value (LSB first), with the following meaning of the bits:

bit 25-32	bit 24	bit 1-23
exponent (exponent bias = 129)	sign (1 = negative)	mantissa

So any number can be calculated using equation 1:

$$number = sign * mantissa * 2^{exponent} \qquad (1)$$

where,

$sign = (-1)^{<bit\ 24>}$
$mantissa = 1 + (<bit\ 1\text{-}23> / 2^{23})$
$exponent = <bit\ 25\text{-}32> - 129$

String
Strings are stored as a stream of bytes using the ASCII encoding (number of bytes according to the file header information).

Memory Dump
Memory dumps are stored as byte values (number of bytes according to the file header information).

Converting Waveform to Bit-Stream

In order to write a tool that is able to convert the waveform into usable data we had to develop a method of interpreting the waveform programmatically and detecting the various stages in the signal.

In our tests the signal was sampled as a 48 kHz, 16 bit, mono signal. As the C7420 outputs the signal at a rate of 1200 bits per second, we can calculate the number of samples per encoded bit (spb) using equation 2:

$$spb = \frac{f}{bps} \qquad (2)$$

where,

spb = samples per bit in the digitized audio stream
f = sample frequency of the waveform
bps = bits per second as output from the C7420

The signal output by the C7420 is a sine wave with a frequency of 4.8 kHz, so every bit is represented by 4 sine periods.

We implemented two different methods of interpreting the signal. Method 1 was taken from the sample programs we got from René van den Enden. For each sample we need to decide if it marks silence or signal. The algorithm scans the sample stream of the digitized waveform until an absolute value greater than half the maximum amplitude of the signal is found. High amplitude is interpreted as a signal and such as the start of a coded bit "1". More samples are subsequently read and counted either as "signal" or "no signal". If more than a certain amount of "no signal" samples are found, it is assumed that the end of a coded "1" has been reached and a coded "0" starts. For a coded "1" bit to be properly recognized, half the number of samples over the duration of 4 sine waves has to be interpreted as "signal". Figure 3 shows a sample waveform and the values counted as "signal" (marked on the horizontal axis as "1") and "no signal" (marked as "0").

While we were able to read the original signal output by the console system without errors using this method, we encountered the following problems when we tried to interpret the signal stored on audio tapes:

- Missing parts of a coded bit: As a certain threshold of "no signal" was defined as the beginning of a coded "0", errors were encountered while interpreting the signal if a small part of the bit had been lost due to data loss on the audio tape.
- Noise in the signal: Most parts of the tapes contained noise which was incorrectly interpreted as signal.
- Differences in amplitude due to independent recordings on the same tape: While we were able to adjust the level of the input signal using the software for recording the signal from the audio source, changes in the signal over various parts of one tape made parts of the tape unreadable.

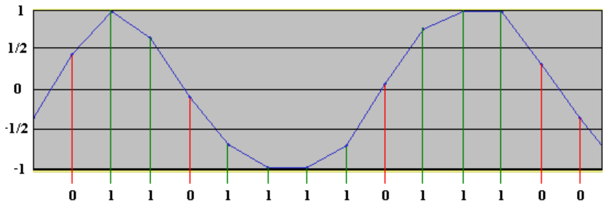
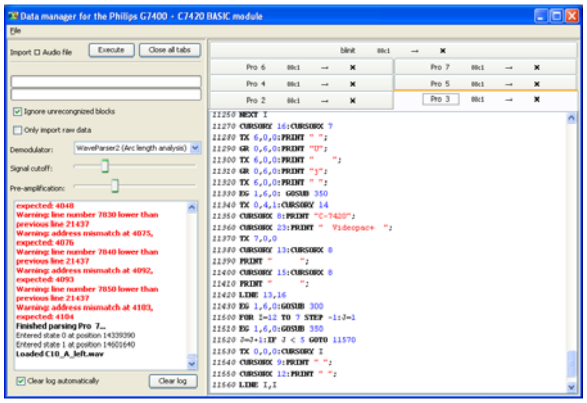


Figure 3: Interpretation of the wave signal using method 1. Vertical axis shows the strength of the amplitude, horizontal axes the parts of the sine wave interpreted as "signal" (1) or "no signal" (0).



Ignore unrecongnized blocks: ✓
Only import raw data: ☐
Demodulator: WaveParser2 (Arc length analysis)
Signal cutoff:
Pre-amplification:

```
expected: 4048
Warning: line number 7830 lower than previous line 21437
Warning: address mismatch at 4075, expected: 4076
Warning: line number 7840 lower than previous line 21437
Warning: address mismatch at 4092, expected: 4093
Warning: line number 7850 lower than previous line 21437
Warning: address mismatch at 4103, expected: 4104
Finished parsing Pro 7...
Entered state 0 at position 14339390
Entered state 1 at position 14601640
Loaded C10_A_left.wav
```

Clear log automatically: ✓

```
11250 NEXT I
11270 CURSORY 16:CURSORX 7
11280 TX 6,0,0:PRINT " ";
11290 GR 0,6,0:PRINT "U";
11300 TX 6,0,0:PRINT "    ";
11310 GR 0,6,0:PRINT "j";
11320 TX 6,0,0:PRINT " ";
11330 EG 1,6,0: GOSUB 350
11340 TX 0,4,1:CURSORY 14
11350 CURSORX 8:PRINT "C-7420";
11360 CURSORX 23:PRINT "  Videopac+  ";
11370 TX 7,0,0
11380 CURSORY 13:CURSORX 8
11390 PRINT "        ";
11400 CURSORY 15:CURSORX 8
11410 PRINT "        ";
11420 LINE 13,16
11430 EG 1,6,0:GOSUB 300
11500 FOR I=12 TO 7 STEP -1:J=1
11510 EG 1,6,0:GOSUB 350
11520 J=J+1:IF J < 5 GOTO 11570
11530 TX 0,0,0:CURSORY I
11540 CURSORX 9:PRINT " ";
11550 CURSORX 12:PRINT " ";
11560 LINE I,I
```

Figure 4: Screenshot of the migration tool GUI with BASIC program loaded from a WAV-file.

To reduce the sensitivity of the algorithm that converts the waveform into a bit stream we implemented a second method. For Method 2 we not only looked at single samples in the waveform but also calculated the sum of piecewise linear approximations of the amplitude, thus calculating the arc length of the sine wave for silence and signal first. Obviously, the arc length of a curve for a bit that represents "1" is longer than the arc length of a curve for a bit that represents "0". To decide if a bit is set or cleared, a cut-off value between signal and silence wave arc length is used.
For every sample in the signal, a certain amount of samples before it are used to calculate the arc length of the sine wave up to the sample. If the arc length is above the cut-off value then the sample is recognized as "1" otherwise it is recognized as "0".

The algorithm is also able to adjust itself to changes in volume or noise, as the threshold which decides if a bit is set or cleared is constantly adjusted for every file in an input stream in parts of the signal which are known to be signal or silence. This way we are able to compensate for noise in the signal as well as for changes in volume. Missing parts of a signal bit have less influence in the recognition as not only the missing part, but also all parts before it are used to decide the state of the bit.

Migration Tool

Using the algorithms for converting the digitized waveform to a binary stream native to the system, together with the information we gathered about file formats, we developed a tool that is able to read the data contained in the waveform. Both described methods of interpreting the waveform were implemented.

The tool is written in JAVA. By using a virtual machine as a platform, the tool is independent from actual hardware for better long term stability. The tool and demo files can be found on the project homepage[6].

The following functions were implemented in the migration tool:

- Opening an audio stream and loading the contained files (either from an audio file (WAV or FLAC) or directly from an audio-in device)
- Opening files in the C7420-native file format (binary streams converted from WAV-file)
- Saving the opened audio stream as a C7420-native file format (binary stream)
- Saving data in a non-obsolete format (screenshots as PNG, basic-programs and arrays as text files, binary data as binary)
- Saving data as an audio stream (either to an audio file (WAV or FLAC) or directly onto the standard audio-out device)
- Opening and saving compressed Zip-archives containing a collection of migrated files
- Creating new files of the different formats in the application (including syntax highlighting for BASIC-programs)

All the data formats used by the C7420 as described in Section *Re-engineering File Formats* are supported by the migration tool.

Every file is opened in a new tab inside the application in an editor that is linked to the file type. The information associated with the file and stored in the file header (native file name, address in memory to load to) can be edited as well. A Screenshot of the migration tool can be seen in Figure 4.

Evaluation

To evaluate the usability of the migration tool, we recorded different programs and other data as output from the original system. The data was recorded as a waveform using Audacity and then converted to user readable data in the migration tool using both implemented methods for converting the waveform. Then the data was loaded back into the original system, both from the recorded audio stream and from a stream reencoded using the migration tool.

The migration tool was able to restore all the data in the

Figure 5: Tapes used for evaluation of migration tool, left upper corner C10 computer cassette, left lower and right Philips FE-I 60 normal position audio tapes.

[6] http://www.ifs.tuwien.ac.at/dp/hc_audio_migration

waveform as output from the original machine with both methods of converting the signal. The original stream outputted by the machine and the re-encoded stream from the migration tool, both gave the same results when the data was loaded back to the original machine. For a clean signal that was not distorted due to age, the migration tool perfectly read and wrote the data from and to the original machine.

Additionally, we acquired three audio-tapes created with the original system approximately 20 years ago from a private archive. Two of the tapes were standard Philips FE*I 60 normal position audio tapes as used for recording music while one was a C-10 computer cassette tape from manufacturer a11 specially manufactured for recording data (Figure 5). The source who recorded the tapes and the contents is not known.

We used a standard HIFI-system as an audio player and the software Audacity to record the audio streams as 44 KHz, 16 bit mono digital signal. The audio streams were saved as uncompressed WAV-files (Petermichl 2009) containing the pulse code modulated (PCM) (Cattermole 1969) raw audio data as bit stream. Two of the tapes had data recorded on both sides of the tape; one had data only on side A. Five WAV-files were obtained, one per side and per tape.

Each file was then loaded using the migration tool. The resulting migrated files were stored in a Zip-archive. For comparison the files were also loaded onto the original system.

A visual check for the characteristic waveform was done using Audacity to see how many files we expected the migration tool and the original system to find. A comparison between expected and loaded files can be found below (first column for each method shows recognized files, second shows unrecognized files and third shows false positives):

tape-side	visual	C7420		method 1		method 2		
C10-A	8	**5**	3	**0**	8	**7**	1	5
C10-B	2	**1**	1	**0**	2	**2**	0	0
Philips-1-A	6	**0**	6	**0**	6	**6**	0	3
Philips-2-A	6	**0**	6	**0**	6	**6**	0	2
Philips-2-B	1	**0**	1	**0**	1	**1**	0	0
Total	**23**	**6**	17	**0**	0	**22**	1	10

Some files on the C10 tape were recognized by the original system but could not be loaded due to a "Bad label" error (with the suggestion to reposition the tape); while on the other 2 tapes no files were recognized at all. No files were correctly recognized using method 1. All but one file were recognized by method 2. Ten additional files recognized using method 2 were false positives that were easily detectable in the user interface and recognition could even be suppressed by checking a checkbox in the migration tool.

The files that were recognized contained BASIC-programs. To check the files for validity we loaded them on the original system from tape and also loaded them on the original system as output from the migration tool.

From the 23 files on the three tapes no file was readable and usable on the C7420. All the 6 files that were recognized on the tapes were loaded with a "Bad file" error message and were not usable due to missing lines and incorrectly interpreted bytes. Thus the original system could not be used to load the data from the original tapes.

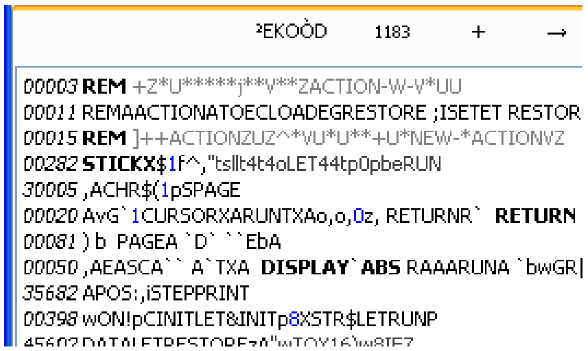
The results of recognized data in the loaded files using the migration tool can be seen below:

tape-side	loaded	not recognized or wrong file format	with errors	no errors
C10-A	7	0	4	3
C10-B	2	0	2	0
Philips-1-A	6	1	5	0
Philips-2-A	6	1	5	0
Philips-2-B	1	1	0	0
Total	22	3	16	3

Of the 22 files loaded 3 files could be recognized without errors. 16 files were loaded with various warnings in the migration tool, indicating that some bytes could not be recognized or were misidentified (e.g. wrong checksum, missing bits in bytes). Three files were not recognized in the correct format and shown as binary stream only.

Without manual preprocessing of the waveform or manual post-processing of the binary stream we were able to recover 19 files opposed to just 6 files loaded by the original system.

The files loaded with errors were in various states of completeness. Some files were missing various lines at the end of the file. Other program lines were erroneous due to incorrectly identified bytes (an example can be seen in Figure 6). As the original data stored on the tapes was not available for comparison, it is not possible to quantify the errors. But in general it seems that only single bytes were lost. As the data on the tapes consists of BASIC programs it should be possible to correct the errors by re-engineering the recovered program sources and thus reconstruct most of the data on the tapes.



```
00003 REM +Z*U*****j**V**ZACTION-W-V*UU
00011 REMAACTIONATOECLOADEGRESTORE ;ISETET RESTOR
00015 REM ]++ACTIONZUZ^*VU*U**+U*NEW-*ACTIONVZ
00282 STICKX$1f^,"tsllt4t4oLET44tp0pbeRUN
30005 ,ACHR$(1pSPAGE
00020 AvG`1CURSORXARUNTXAo,o,0z, RETURNR`  RETURN
00081 ) b  PAGEA `D` ``EbA
00050 ,AEASCA`` A`TXA  DISPLAY`ABS RAAARUNA `bwGR|
35682 APOS:,iSTEPPRINT
00398 wON!pCINITLET&INITp8XSTR$LETRUNP
45602 DATALETRESTOREzA"wTOY16)w8IEZ
```

Figure 6: Errors in BASIC-Program decoded from a file loaded using the migration tool.

Conclusions

The case study performed in this work proved that it is possible to extract proprietary data from the analog audio signal stored by a system without previous knowledge of the format it is stored in. By having access to the original system to write test programs we were able to reengineer the audio waveform as well as all data formats and write a tool to migrate the data to non-obsolete formats. Archives or libraries that have or may receive audio tapes containing data for the Philips G7400 can use this tool to migrate the digital data without access to the original system or knowledge of how to handle the system.

If such actions are undertaken today, while the original systems still work, it is possible to develop tools for the migration of digital objects now. Once the original systems do not work anymore, it will not be possible to run code on the original system, thus having to reengineer the system on a circuit-diagram level and disassembling the BIOS source code, which makes the task more difficult and time consuming.

Reengineering of the system
While digital archeology and reengineering systems is seen as a rather complex task, this case study shows that the reengineering of the format is easier while having access to the original system, as this way, test data can be produced. Interpreting the number format without seeing the effects of the changed numbers on the original machine would have been a rather difficult task. It should also be noted that non-commercial ‘retro gaming’ communities still working with the system can be an excellent source not only for emulation, but also for data archeology on home computer systems.

Information lost due to migration
While most of the information that can be stored in files on the original system can be migrated to non-obsolete formats, certain restrictions apply:

- Screenshots: The G7400 is able to render blinking information on screen. By choosing PNG as a non-obsolete (static) format, this dynamic information of the data is lost. Additionally it is possible to define custom characters using the BASIC language. The definition of these characters is not stored in the waveform with the screenshot data. A complete program with the definition of the custom characters would have to be stored and preserved to keep the information available (e.g. through emulation)
- String Arrays: As a string array contains only the addresses of the strings stored in it and the strings themselves are each stored in separate files, the interrelation between these files is lost without the logic of the program that establishes the link between them.

Evaluated tapes
Examination of the data on tapes from a private archive showed that the data was no longer readable on the original machine. Using the migration tool we were able to retrieve most of the data with small errors. The evaluation also showed that it is necessary to act now and migrate data that was stored on magnetic tapes 20 years ago, as the lifetime of magnetic tapes is expected to be a maximum of 20 years (Van Bogart 1998). Most of the data retrieved in the experiment could not be extracted without errors.

Media refresh
Using the developed migration tool it is possible to refresh the media (audio cassettes) by reading and decoding the content, recoding it into a waveform and recording it to the tape again without using the original system.

Using decoded data for emulation
With the possibility to save the data encoded in the waveform as a system native binary stream, files can be stored for usage in emulators. Currently no emulators for the C7420 are available, but by storing the streams in the native format the data is kept safe for emulation at a future date.

Interpreting results for other media types
As audio tapes can be read using standard non-proprietary audio equipment, access to the physical layer of data is not in immediate danger. Other magnetic media like floppy discs cannot be read as easily. Even with floppy drives using the same media size (8”, 5¼ “, 3½”) access to data written on non-compatible computer systems is not possible.

The results of re-engineering the logical data can be used for other media as well. Re-engineering file formats can either be done using original systems or emulators, if available. Expert knowledge in handling the system has to be at hand to complete these tasks.

Acknowledgements

Part of this work was supported by the European Union in the 6th Framework Program, IST, through the PLANETS project, contract 033789.

References

Bhushan, B. (1992) *Mechanics and Reliability of Flexible Magnetic Media*. Springer, New York.

Cattermole, K. W. (1969) *Principles of Pulse Code Modulation*. Iliffe Books. London, U.K.

Guttenbrunner, M., Becker, C., & Rauber, A. (2008, September). Evaluating strategies for the preservation of console video games. In *Proceedings of the Fifth international Conference on Preservation of Digital Objects* (iPRES 2008), London, UK, pp. 115–121.

Matthews, B., McIlwrath, B., Giaretta, D., and Conway, E. 2008. *The significant properties of software: A study*. JISC Study.
Retrieved September 1, 2009, from http://www.jisc.ac.uk/media/documents/programmes/preservation/spsoftware_report_redacted.pdf.

Petermichl, K. (2009) *Handbuch der Audiotechnik: Kapitel 12: Dateiformate für Audio*. Springer Berlin Heidelberg, Germany.

Ross, S., and Gow, A. (1999). *Digital archaeology: Rescuing neglected and damaged data resources. A JISC/NPO study within the Electronic Libraries (eLib) Programme on the Preservation of Electronic Materials*. Retrieved September 1, 2009, from http://eprints.erpanet.org/47/.

Rothenberg, J. (2000). *Using Emulation to Preserve Digital Objects.* Koninklijke Bibliotheek. Retrieved September 1, 2009, from http://www.kb.nl/pr/publ/usingemulation.pdf

Van Bogart, J. (1998) *Storage Media Life Expectancies.* Digital Archive Directions (DADs) Workshop 1998. Retrieved September 1, 2009, from http://nost.gsfc.nasa.gov/isoas/dads/presentations/VanBogart/

Webb, C. (2005). *Guidelines for the Preservation of the Digital Heritage*. Information Society Division United Nations Educational, Scientific and Cultural Organization (UNESCO) – National Library of Australia. Retrieved May 29, 2009, from http://unesdoc.unesco.org/images/0013/001300/130071e.pdf

IEEE (1987). *IEEE Standard 754-1985 for Binary Floating Point Arithmetic*, IEEE, (1985). Reprinted in SIGPLAN 22(2) pp. 9-25.