PARADOX CAT

IN CAR GRAPHICS TECHNOLOGIES

# Programming Test

# Decoding Exercise

“Would you hire a magician without asking them to show you some magic tricks? Of course not.” - The Joel Test: 12 Steps to Better Code

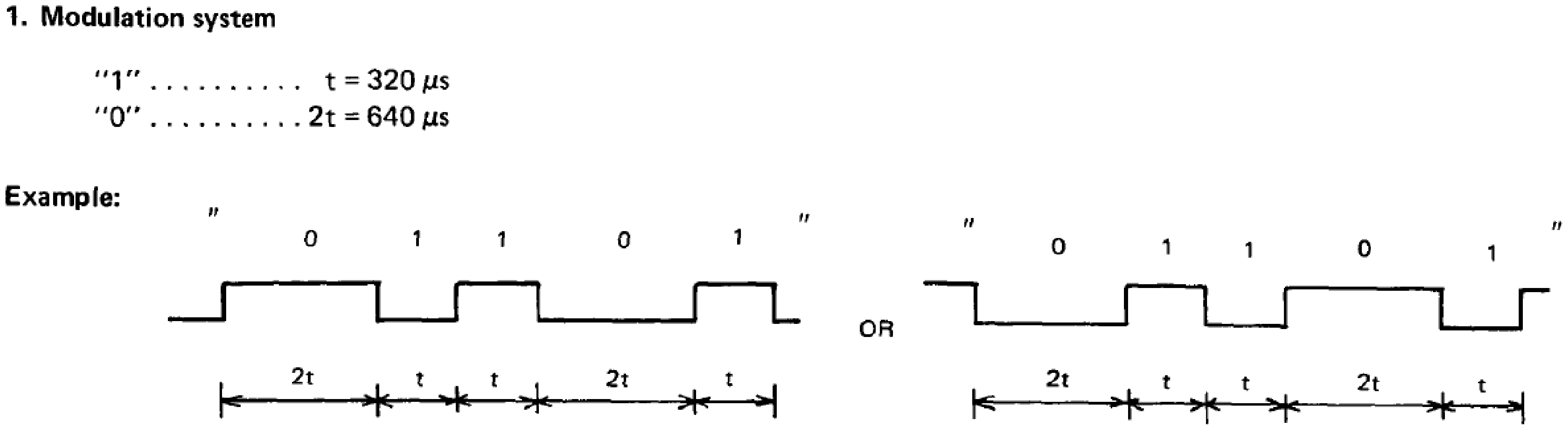
# Approach

- **We strongly believe that programming is fun, and we hope you will find this a fun assignment as well!**
- **There is nothing better to have some harmless fun and challenge than to do some real retro-programming!**
- **Use the programming language you feel most comfortable with in creating a solution for the problem.**
- **Of course, find a system function or library to read the data file for you.**
- **Don‘t dive too deep into the AFSK explanations in the web – the encoding is actually pretty simple.**
- **Feel free to introduce any other helpful library functions that makes your job easier. It is not really necessary, but helps keeping the code on a higher level.**
- **Time-box yourself – this is a hard assignment and it is not required to finish to show your effort.**
- **This is to allow us to talk about your approach to an unknown problem.**
- **What tool beside your IDE do you think useful for this type of problem?**

# Instructions

- **Given the audio file in WAV format (contained in the ZIP archive together with this instructions), decode the binary data encoded in it.**
- **The data is encoded using Audio Frequency Shift-Keying (AFSK) in its simplest form**
  - **A single bit is the waveform between two zero-crossings**
  - **A one signal is a rectangle signal of t = 320 microseconds**
  - **A zero signal is a rectangle signal of t = 640 microseconds**
  - **The real-life data might no longer be an ideal rectangle, since it has undergone storage on physical media (e.g. a tape drive)**
- **The bit-stream that can be extracted from the decoded audio signal can be converted into bytes**
  - **The signal starts with a lead tone of roughly 2.5 seconds (all 1-bits, or 0xff bytes), and ends with an end block of about 0.5 seconds (all 1-bits).**
  - **11 bits are used to encode a single byte – 8 bits for the byte plus one start bit (valued 0) and two stop bits (valued 1)**
  - **The data is encoded with least-significant bit first**
- **The byte-stream has the following form:**
  - **The first two bytes are 0x42 and 0x03**
  - **After that, construct 64 messages of 30 bytes each, with the 31st byte being the checksum of the 30 bytes before that (you need 1984 bytes = 64 * 31 for that)**
  - **The last byte before the end block is a 0x00 byte.**
- **The checksums will help you detect that your encoding works**
- **The data in this real-life file will have no meaning to you, unless you figure out which machine it was created by – this could be near impossible (don’t try).**

# Explanation: Binary encoding



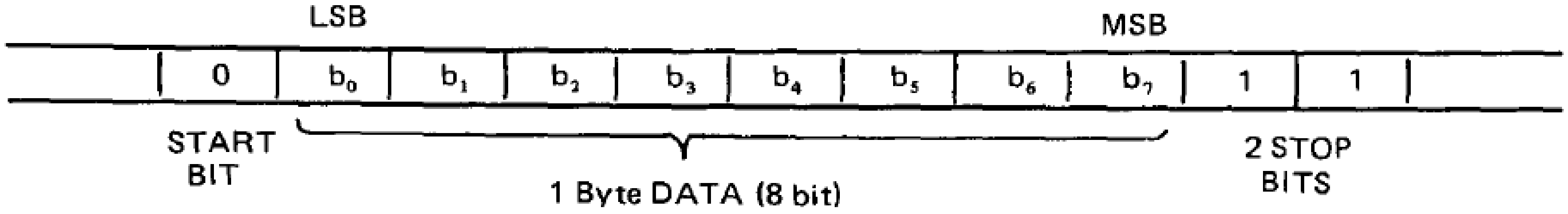
**1. Modulation system**

"1" . . . . . . . . . . t = 320 $\mu$s
"0" . . . . . . . . . . 2t = 640 $\mu$s

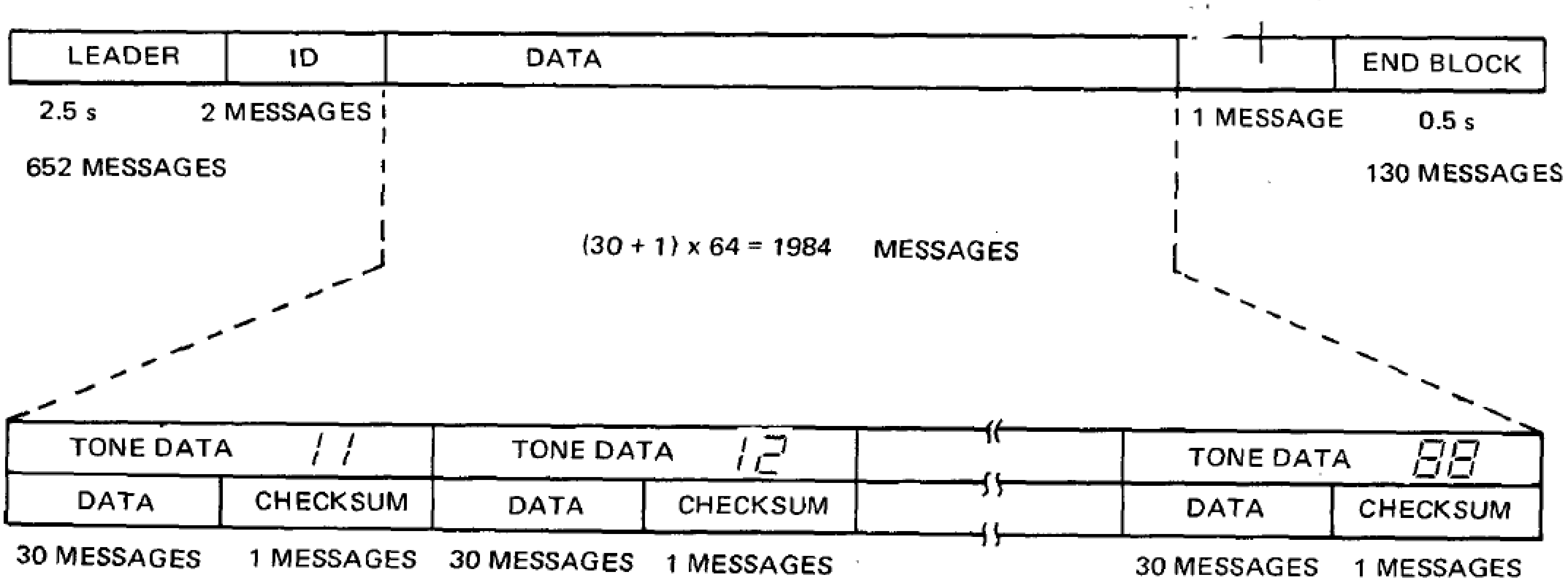
**Example:**

" 0 1 1 0 1 "
2t
t
t
2t
t
OR
" 0 1 1 0 1 "
2t
t
t
2t
t

# Explanation: Bit-stream encoding, a single byte in the bit-stream



# Explanation: Message format, overall structure and checksum positions in byte-stream


LEADER
ID
DATA
END BLOCK
2.5 s
2 MESSAGES
652 MESSAGES
1 MESSAGE
0.5 s
130 MESSAGES
(30 + 1) x 64 = 1984 MESSAGES
TONE DATA 11
TONE DATA 12
TONE DATA 88
DATA
CHECKSUM
DATA
CHECKSUM
DATA
CHECKSUM
30 MESSAGES
1 MESSAGES
30 MESSAGES
1 MESSAGES
30 MESSAGES
1 MESSAGES