Documentation for the Assignment

Omkar Shirpure (22B0910)
Suryansh Patidar (22B1036)
Krish Rakholiya (22B0927)
Parthiv Sen (22B1055)

1 Introduction

This document provides a detailed explanation of the scripts `sender-combined.py` and `receiver-combined.py`. These scripts are designed to simulate the encoding, transmission, reception, and error detection of binary data using audio signals. The process involves encoding data with CRC (Cyclic Redundancy Check), simulating transmission errors, and decoding the received data to detect and identify any corrupted bits.

2 Sender Script: `sender-combined.py`

2.1 Function Overview

2.1.1 `xor(a, b)`

Description: Performs the XOR operation between two binary strings.
Input:

- `a`: A binary string (e.g., "10101").
- `b`: Another binary string of the same length as `a`.

Output: A binary string resulting from the XOR operation.

2.1.2 `mod2div(dividend, divisor)`

Description: Performs modulo-2 division (bitwise XOR division) to calculate the remainder. This is typically used in CRC encoding.
Input:

- `dividend`: A binary string representing the data to be divided.
- `divisor`: A binary string representing the divisor (CRC key).

Output: The remainder of the modulo-2 division.

2.1.3 `encode_data(data, key)`

Description: Encodes the given binary string (`data`) using the provided CRC key (`key`).
Input:

- `data`: A binary string to be encoded.
- `key`: The CRC key (binary string) used for encoding.

Output: The encoded data, which includes the original data appended with the CRC remainder.

2.1.4 `flip_bits(data, positions)`

Description: Flips the bits at the specified positions in the binary string.
Input:

- `data`: A binary string in which certain bits are to be flipped.
- `positions`: A list of integer indices where the bits should be flipped.

Output: The binary string after flipping the specified bits.

2.1.5 `transmit(bitstring)`

Description: Simulates the transmission of the binary string by converting each bit into an audio signal of a specific frequency. Binary 0 is transmitted as a 5000 Hz tone, while binary 1 is transmitted as a 7000 Hz tone.
Input:

- `bitstring`: The binary string to be transmitted as an audio signal.

Output: None. The function directly transmits the audio signal.

2.1.6 `inputData()`

Description: Prompts the user to input a binary string.
Input: None (takes input from the user interactively).
Output: The user-provided binary string.

2.2 Execution Flow

1. The program starts by calling `inputData()` to get the binary string (`data`) from the user.
2. The data is then encoded using the CRC key via `encode_data()`.
3. The encoded data is printed, and the user is asked to specify up to two bit positions to flip in the encoded data using `flip_bits()`.
4. The encoded data is length-prefixed, and if bit flipping is requested, the bit positions are adjusted considering the length prefix.
5. The corrupted data is length-prefixed and then transmitted as an audio signal using `transmit()`.
6. The elapsed time for the entire process is printed at the end.

3 Receiver Script: `receiver-combined.py`

3.1 Function Overview

3.1.1 `CRC(dataword, generator)`

Description: Computes the CRC (Cyclic Redundancy Check) for a given dataword using a specified generator polynomial.
Input:

- `dataword`: A binary string representing the data to be checked.
- `generator`: A binary string representing the generator polynomial.

Output: The remainder of the CRC computation.

3.1.2 `encode(dataword, generator)`

Description: Encodes the given dataword by appending the CRC remainder to it.
Input:

- `dataword`: A binary string to be encoded.
- `generator`: The CRC key (binary string) used for encoding.

Output: The encoded binary string.

3.1.3 `checkError(codeword, generator)`

Description: Checks if the given codeword contains any errors using the CRC generator.
Input:

- `codeword`: The received binary string to be checked.
- `generator`: The CRC key (binary string) used for error detection.

Output: Boolean indicating whether the codeword is error-free.

3.1.4 `flipBitsAt(codeword, error_positions)`

Description: Flips the bits at specified positions in the binary string.
Input:

- `codeword`: A binary string in which certain bits are to be flipped.
- `error_positions`: A list of integer indices where the bits should be flipped.

Output: The binary string after flipping the specified bits.

3.1.5 `evaluate(dataword, generator)`

Description: Attempts to detect and correct errors in the received dataword by checking different bit positions.
Input:

- `dataword`: The received binary string to be evaluated.
- `generator`: The CRC key (binary string) used for error detection.

Output: A dictionary containing the results of the evaluation, including detected error positions and time taken.

3.1.6 `decode(datastring)`

Description: Decodes the received data by detecting and reporting any corrupted bits.
Input:

- `datastring`: The binary string received from the audio signal.

Output: The results of the error detection process.

3.1.7 `receive_data()`

Description: Receives and processes an audio signal, filtering it to extract the transmitted bitstring, which is then decoded to detect any errors.
Input: None.
Output: The extracted bitstring.

3.2 Execution Flow

1. The program starts by receiving and processing an audio signal through `receive_data()`.
2. The extracted bitstring is processed to determine the length of the dataword.
3. The relevant portion of the bitstring is passed to `decode()` to detect any corrupted bits.
4. The results, including the corrupted bits and time taken for detection, are printed to the console.

4 Conclusion

The `sender-combined.py` and `receiver-combined.py` scripts work together to simulate a basic data transmission system using audio signals. The sender script encodes and transmits the data, while the receiver script decodes and checks for errors. This setup provides a practical demonstration of CRC-based error detection in a network-like environment.