Opis mini-projektu "Modem dźwiękowy"

1. Opis pliku "generator.py".

Kod, który znajduje się w tym pliku najpierw koduje podane informacje (tzn. adresata, odbiorcę oraz wiadomość) w ramkę, a następnie emituje dwa dźwięki odpowiadające bitowi zapalonemu lub zgaszonemu w ramce, którą wcześniej zakodował.

2. Jak uruchomić program "generator.py" ?

Program przyjmuje jako argumenty 3 zmienne: czas , częstotliwość odpowiadającą bitowi zgaszonemu oraz częstotliwość odpowiadającą bitowi zapalonemu.

Po uruchomieniu program będzie czekał na podanie na wejściu standardowym danych do zakodowania, czyli numer adresata, numer odbiorcy oraz wiadomość.

Przykład:

$ python generator.py 5 440 880

1 3 msg

3. Omówienie kodu "generator.py".

Na początku są zaimportowane wszystkie potrzebne biblioteki do działania programu.

Linia 13: zdefiniowanie zmiennej "amplitude", która odpowiada za głośność emitowanego dźwięku.

Linia 14: wczytanie do zmiennych argumentów podanych w linii komend przy uruchomieniu programu.

Linia 18: stworzenie słownika "tocode", który pomoże przy kodowaniu 4b5b.

Linia 36: definicja funkcji "code4b5b", która odpowiada za kodowanie 4b5b. W pętli while wycinam 4 pierwsze bity podanego stringa, zamieniam je odpowiednio ze słownikiem "tocode" i dodaje do stringa wyjściowego.

Linia 42: definicja funkcji "codenrz", która odpowiada za kodowanie NRZ. Pierwszy "poprzedni" bit ustawiam na 1. W pętli while wycinam pierwszy bit podanego stringa, sprawdzam, czy różni się od poprzedniego bitu, jeżeli tak to zmieniam na przeciwny bit.

Linia 54: definicja funkcji "encoding", w której zamieniam adresata, odbiorcę oraz wiadomość na kod binarny oraz liczę CRC powstałego stringa. Następnie korzystam z funkcji "code4b5b" oraz "codenrz" oraz dodaję preambułę.

Linia 67: definicja funkcji "generplay", która generuje potrzebne dźwięki. Najpierw liczę dwie sinuoidy odpowiednio dla freq0 oraz freg1. Następnie inicjuję playera i iteruję się po bitach, które są wynikiem kodowania i odpowiednio generuje ton dla zera lub jedynki.

4. Opis pliku "reader.py".

Kod, który znajduje się w tym pliku, nasłuchuje dźwięk i jeżeli dane są poprawne dekoduje otrzymaną wiadomość.

5. Jak uruchomić program "reader.py"?

Program przyjmuje jako argumenty 3 zmienne: czas , częstotliwość odpowiadającą bitowi zgaszonemu oraz częstotliwość odpowiadającą bitowi zapalonemu.

Przykład:

$ __PULSEAUDIO_WAVFILE__=jakisplik.wav python reader.py 5 440 880

6. Omówienie kodu "reader.py".

Na początku są zaimportowane wszystkie potrzebne biblioteki do działania programu.

Linia 8: definicja zmiennych, z których później kod będzie korzystał.

Linia 10: Linia 18: stworzenie słownika "todecode", który pomoże przy dekodowaniu 4b5b.

Linia 29: wczytanie do zmiennych argumentów podanych w linii komend przy uruchomieniu programu.

Linia 33: definicja funkcji "dominatfreq", która nasłuchuje przez czas podany jako argument "time" i zwraca dominującą częstotliwość. Jeżeli długość tablicy "tab" jest równa 0 to znaczy, że nic nie wczytaliśmy, więc zwracamy błąd.

Linia 41: definicja funkcji "decodenrz", która dekoduje kod NRZ. Pierwszy "poprzedni" bit ustawiam na 1. W pętli while wycinam pierwszy bit podanego stringa, sprawdzam, czy różni się od poprzedniego bitu, jeżeli tak to dodaje do stringa wyjściowego "1", jeżeli nie to "0".

Linia 54: definicja funkcji "decode4b5b", w której kolejne 5 bitów podanego stringa dekoduję na 4 bity odpowiednio ze słownikiem "todecode" i dodaje do zwracanego stringa.

Linia 62: definicja funkcji "decode", która podany string dekoduje za pomocą funkcji "decodenrz" oraz "decode4b5b". W uzyskanym stringu zamieniam bity od 48-95 na liczbę odpowiadającą adresatowi, bity od 0 do 47 na liczbę odpowiadającą odbiorcy i zapisuje do stringu wyjściowego.
Z bitów od 96 do 111 oczdczytuje długość zakodowanej wiadomości. W zmiennej endofmsg znajduje się ostatni bit, na którym zakodowana jest wiadomość. Z bitów od 112 do endofmsg dekoduję treść wiadomości i dopisuję ją do stringu wyjściowego.

Linia 72: definiuję rekorder.

Linia 77: Następnie definiuję zmienną "err", która będzie kończyć działanie pętli while (linia 74), jeżeli pojawi się jakiś błąd.

Linia 76: while omija dzwięki, które nie są dla nas istotne i są tylko jakimiś szmerami.

Linia 83: Podczas słuchania preambuły synchronizuję się. Podczas 5 prób szukam indexu, dla którego częstotliwość jest największa, czyli jedynki.

Jeżeli jest większa to nadpisuję wartość szukanego indexu, a jeżeli mniejsza niż częstotliwość dla zera to kończę głównego while, ponieważ słychać dźwięki, które nas nie interesują. Po każdej z 5 prób

przesuwam "ramkę"(nframes) o 1/5 długości, oraz jeżeli pętla for nie została przerwana breakiem to przesuwam "nframes" o index/5 długości.

Linia 101: w tej pętli while czekam, aż wystąpią dwie jedynki obok siebie, co znaczy, że zakończyła się preambuła. Zatem, można przejść do słuchania wiadomości.

Linia 110: w pętli while nasłuchuję dominującą częstotliwość przez funkcję "dominatfreq", następnie zwracane dane przez funkcję "dominatfreq" dopisuję do stringa, którego następnie zdekoduję przy użyciu funkcji "decode", a następnie wypiszę na konsolę.