Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Основы компьютерных сетей

ОТЧЕТ

по лабораторной работе №4

на тему

СЛУЧАЙНЫЕ МЕТОДЫ ДОСТУПА К МОНОКАНАЛУ

Студент М.С. Патюпин

Преподаватель В.А. Марцинкевич

МИНСК 2024

**1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

**1.1 Исходные данные**

Для написания и отладки программы использовалась среда разработки Visual Studio Code. В качестве языка программирования был выбран язык Python и библиотека tkinter для создания графического интерфейса (GUI).

Для эмуляции COM-портов использовался Virtual Serial Ports Driver SDK, задействующий архитектуру UART 16550.

Взять за основу программу, относящуюся к лабораторной работе №3.

**1.2 Вариант задания**

Нечётный вариант.

На стороне передатчика, реализовать три ключевых шага алгоритма: прослушивание канала, обнаружение коллизии и розыгрыш случайной задержки (в соответствующей последовательности).

Предусмотреть возможность эмуляции занятости канала. Вероятность занятости канала должна составлять 50 %.

Коллизию рассматривать применительно к кадру целиком (не к байту). Вероятность коллизии должна составлять 60%.

Для расчета случайной задержки использовать стандартную формулу.

Реализовать поддержку jam-сигнала (дополнительно и правильно; как на стороне передатчика, так и на стороне приемника).

**1.2 Теоретические сведенья**

Алгоритм CSMA/CD (Carrier Sense Multiple Access with Collision Detection) представляет собой метод управления доступом к общей среде передачи данных в компьютерных сетях. Этот протокол широко применяется в локальных сетях Ethernet и позволяет нескольким устройствам эффективно использовать один канал связи.

Основной принцип работы CSMA/CD заключается в том, что устройства, желающие передать данные, сначала "прослушивают" канал, чтобы убедиться, что он свободен. Если канал свободен, устройство начинает передачу. В случае, если два или более устройств начинают передачу одновременно, происходит коллизия. Устройства способны обнаружить эту коллизию и прекращают передачу. После этого они ждут случайный промежуток времени перед повторной попыткой передачи, что снижает вероятность повторного возникновения коллизии.

В контексте передачи данных через COM-порты, упрощенный алгоритм CSMA/CD адаптируется к особенностям последовательной связи. COM-порты обеспечивают двунаправленную связь между компьютером и периферийными устройствами, и применение принципов CSMA/CD позволяет организовать эффективный обмен данными при наличии нескольких устройств, подключенных к одной линии связи.

Пакетная передача данных в этом контексте предполагает разделение информации на небольшие блоки (пакеты), каждый из которых содержит служебную информацию (заголовок) и полезные данные. Такой подход позволяет лучше управлять потоком данных, обнаруживать и исправлять ошибки, а также эффективно использовать канал связи.

**1.3 Алгоритм CSMA/CD на стороне передатчика**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**1.4 Алгоритм CSMA/CD на стороне приемника**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**2 ПРАКТИЧЕСКАЯ ЧАСТЬ**

**2.1 Код программы**

Код программы вынесен в приложение А.

**2.2 Интерфейс программы**

Интерфейс программы представлен на рисунке 2.1.

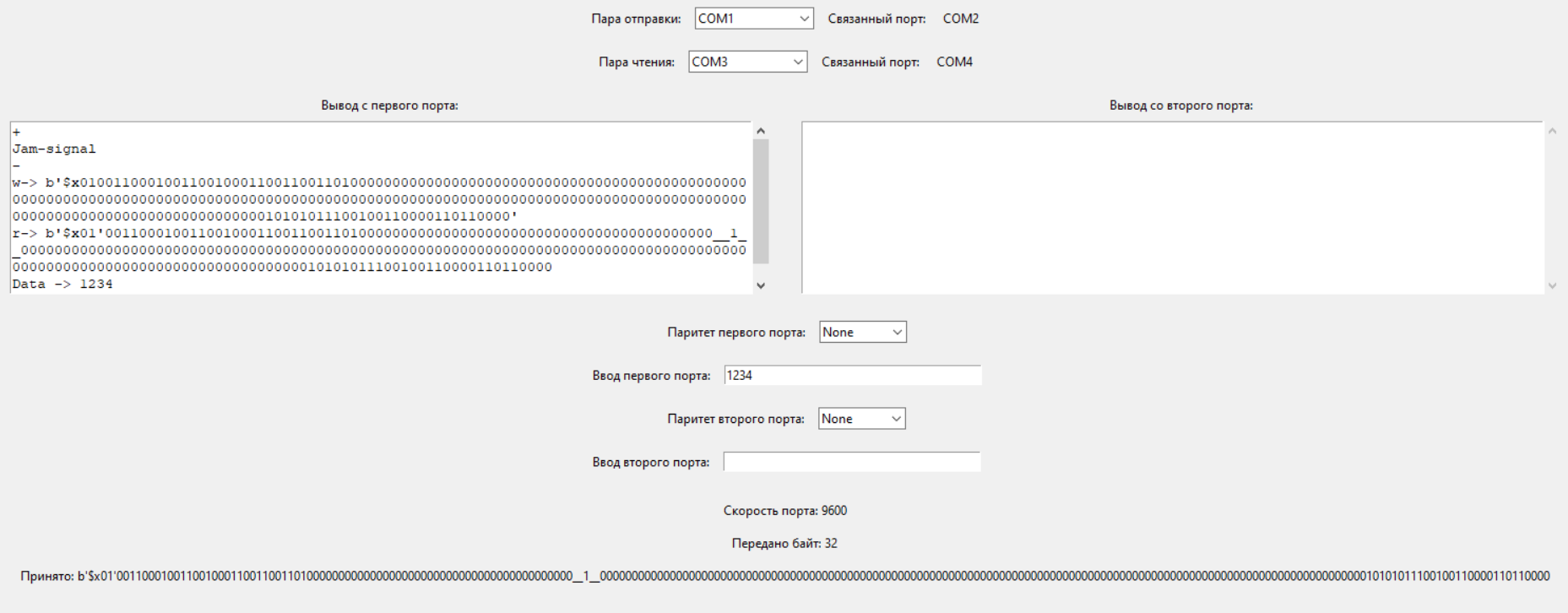


Рисунок 2.1 – Интерфейс программы

Приложение А

*(Код программы)*

Файл makefile

1. PYTHON=python
2. SRC\_DIR=src
3. INTERFACE\_FILE=$(SRC\_DIR)/serial\_interface.py
4. LOGIC\_FILE=$(SRC\_DIR)/serial\_logic.py
5. MAIN\_FILE=main.py

6.

1. .PHONY: run clean
2. run:

10. $(PYTHON) $(MAIN\_FILE)

11.

12. clean:

13. del /S \*.pyc

Файл main.py

1. import sys

2. import os

3. sys.path.insert(0, os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), 'src')))

4. from serial\_interface import SerialInterfaceApp

5. import tkinter as tk

6. if \_\_name\_\_ == "\_\_main\_\_":

7. root = tk.Tk()

8. app = SerialInterfaceApp(root)

9. root.protocol("WM\_DELETE\_WINDOW", app.on\_closing) # Обрабатываем закрытие окна

10. root.mainloop()

Файл serial\_logic.py

1. import serial

2. import serial.tools.list\_ports

3. import time

4. import threading

5. import random

6. class SerialLogic:

7. def \_\_init\_\_(self):

8. self.port\_pairs = {}

9. self.stop\_event = threading.Event()

10. self.read\_threads = []

11. self.port\_speeds = {}

12. self.baudrate = 9600

13. def get\_port\_speed(self, port):

14. return self.baudrate

15. def populate\_ports(self):

16. ports = serial.tools.list\_ports.comports()

17. self.ports\_list = [port.device for port in ports if port.device not in ["COM6", "COM7", "COM15", "COM16"]]

18. if len(self.ports\_list) < 4:

19. raise ValueError("Недостаточно доступных COM-портов.")

20. self.create\_port\_pairs()

21. return self.ports\_list

22. def create\_port\_pairs(self):

23. checked\_ports = set()

24. for i in range(len(self.ports\_list)):

25. port1 = self.ports\_list[i]

26. if port1 in checked\_ports:

27. continue

28. for j in range(i + 1, len(self.ports\_list)):

29. port2 = self.ports\_list[j]

30. if port2 in checked\_ports:

31. continue

32. if self.test\_connection(port1, port2):

33. self.port\_pairs[port1] = port2

34. self.port\_pairs[port2] = port1

35. checked\_ports.add(port1)

36. checked\_ports.add(port2)

37. print(f"Связь установлена: {port1} <-> {port2}")

38. break

39. def test\_connection(self, port1, port2):

40. print(f"Тестируем соединение между {port1} и {port2}...")

41. try:

42. with serial.Serial(port1, self.baudrate, timeout=1) as ser1, serial.Serial(port2, self.baudrate, timeout=1) as ser2:

43. test\_message = "TEST"

44. ser1.write(test\_message.encode())

45. if ser2.in\_waiting > 0:

46. response = ser2.read(ser2.in\_waiting).decode()

47. print(f"Получено сообщение: {response} от {port2}")

48. return response == test\_message

49. else:

50. print("Нет ответа от порта.")

51. return False

52. except serial.SerialException as e:

53. print(f"Ошибка: {e}")

54. return False

55. \_

56. \_

57. \_

58. \_

59. \_

60. \_

61. \_

62. \_

63. def send\_packet(self, send\_port, receive\_port, message, parity):

64. parity\_setting = {

65. "None": serial.PARITY\_NONE,

66. "Even": serial.PARITY\_EVEN,

67. "Odd": serial.PARITY\_ODD

68. }[parity]

69. self.port\_speeds[send\_port] = self.baudrate

70. self.port\_speeds[receive\_port] = self.baudrate

71. try:

72. with serial.Serial(send\_port, self.baudrate, timeout=1, parity=parity\_setting) as ser1, \

73. serial.Serial(receive\_port, self.baudrate, timeout=1, parity=parity\_setting) as ser2:

74. time.sleep(1)

75. ser1.write(message)

76. if ser2.in\_waiting > 0:

77. response = ser2.read(ser2.in\_waiting)

78. print(f"r-> {response}")

79. return response

80. else:

81. print("Нет ответа от порта.")

82. return None

83. except serial.SerialException as e:

84. print(f"Ошибка: {e}")

85. return None

Файл serial\_interface.py

1. import tkinter as tk

2. \_

3. from tkinter import scrolledtext, ttk, messagebox

4. from serial\_logic import SerialLogic

5. from crc\_code.systematic\_cyclic\_code import SystematicCyclicCode

6. class SerialInterfaceApp:

7. FLAG\_SIZE = 2

8. flag = "$x"

9. DESTINATION\_ADDRESS\_SIZE = 1

10. destination = bytes([0x30])

11. SOURCE\_ADDRESS\_SIZE = 1

12. DATA\_SIZE = 24

13. FCS\_SIZE = 4

14. def \_\_init\_\_(self, master):

15. self.master = master

16. self.master.title("Serial Interface")

17. self.logic = SerialLogic()

18. self.master.minsize(1500, 600)

19. self.setup\_ui()

20. self.populate\_ports()

21. def setup\_ui(self):

22. self.send\_frame = tk.Frame(self.master)

23. self.send\_frame.pack(pady=10)

24. self.send\_port\_label = tk.Label(self.send\_frame, text="Пара отправки:")

25. self.send\_port\_label.pack(side=tk.LEFT, padx=5)

26. self.send\_port\_combobox = ttk.Combobox(self.send\_frame, width=15)

27. self.send\_port\_combobox.pack(side=tk.LEFT, padx=5)

28. self.send\_connected\_port\_label = tk.Label(self.send\_frame, text="Связанный порт:")

29. self.send\_connected\_port\_label.pack(side=tk.LEFT, padx=5)

30. self.send\_connected\_port = tk.Label(self.send\_frame, text="")

31. self.send\_connected\_port.pack(side=tk.LEFT, padx=5)

32. self.send\_port\_combobox.bind("<<ComboboxSelected>>", self.update\_connected\_send\_port)

33. self.read\_frame = tk.Frame(self.master)

34. self.read\_frame.pack(pady=10)

35. self.read\_port\_label = tk.Label(self.read\_frame, text="Пара чтения:")

36. self.read\_port\_label.pack(side=tk.LEFT, padx=5)

37. self.read\_port\_combobox = ttk.Combobox(self.read\_frame, width=15)

38. self.read\_port\_combobox.pack(side=tk.LEFT, padx=5)

39. self.read\_connected\_port\_label = tk.Label(self.read\_frame, text="Связанный порт:")

40. self.read\_connected\_port\_label.pack(side=tk.LEFT, padx=5)

41. self.read\_connected\_port = tk.Label(self.read\_frame, text="")

42. self.read\_connected\_port.pack(side=tk.LEFT, padx=5)

43. self.read\_port\_combobox.bind("<<ComboboxSelected>>", self.update\_connected\_read\_port)

44. self.output\_frame = tk.Frame(self.master)

45. self.output\_frame.pack(pady=10, fill=tk.X)

46. self.left\_output\_frame = tk.Frame(self.output\_frame)

47. self.left\_output\_frame.pack(side=tk.LEFT, padx=10, fill=tk.BOTH, expand=True)

48. self.right\_output\_frame = tk.Frame(self.output\_frame)

49. self.right\_output\_frame.pack(side=tk.RIGHT, padx=10, fill=tk.BOTH, expand=True)

50. self.left\_output\_label = tk.Label(self.left\_output\_frame, text="Вывод с первого порта:")

51. self.left\_output\_label.pack()

52. self.left\_output\_text = scrolledtext.ScrolledText(self.left\_output\_frame, width=100, height=20, state='disabled')

53. self.left\_output\_text.pack(padx=5, pady=5, fill=tk.BOTH, expand=True)

54. self.right\_output\_label = tk.Label(self.right\_output\_frame, text="Вывод со второго порта:")

55. self.right\_output\_label.pack()

56. self.right\_output\_text = scrolledtext.ScrolledText(self.right\_output\_frame, width=100, height=20, state='disabled')

57. self.right\_output\_text.pack(padx=5, pady=5, fill=tk.BOTH, expand=True)

58. self.parity\_frame\_send = tk.Frame(self.master)

59. self.parity\_frame\_send.pack(pady=10)

60. self.parity\_label\_send = tk.Label(self.parity\_frame\_send, text="Паритет первого порта:")

61. self.parity\_label\_send.pack(side=tk.LEFT, padx=5)

62. self.parity\_combobox\_send = ttk.Combobox(self.parity\_frame\_send, values=["None", "Even", "Odd"], state='readonly', width=10)

63. self.parity\_combobox\_send.current(0)

64. self.parity\_combobox\_send.pack(side=tk.LEFT, padx=5)

65. self.input\_frame\_send = tk.Frame(self.master)

66. self.input\_frame\_send.pack(pady=10)

67. self.input\_label\_send = tk.Label(self.input\_frame\_send, text="Ввод первого порта:")

68. self.input\_label\_send.pack(side=tk.LEFT, padx=5)

69. self.input\_entry\_send = tk.Entry(self.input\_frame\_send, width=40)

70. self.input\_entry\_send.pack(side=tk.LEFT, padx=5)

71. self.input\_entry\_send.bind("<Return>", self.send\_message\_first)

72. self.parity\_frame\_receive = tk.Frame(self.master)

73. self.parity\_frame\_receive.pack(pady=10)

74. self.parity\_label\_receive = tk.Label(self.parity\_frame\_receive, text="Паритет второго порта:")

75. self.parity\_label\_receive.pack(side=tk.LEFT, padx=5)

76. self.parity\_combobox\_receive = ttk.Combobox(self.parity\_frame\_receive, values=["None", "Even", "Odd"], state='readonly', width=10)

77. self.parity\_combobox\_receive.current(0)

78. self.parity\_combobox\_receive.pack(side=tk.LEFT, padx=5)

79. self.input\_frame\_receive = tk.Frame(self.master)

80. self.input\_frame\_receive.pack(pady=10)

81. self.input\_label\_receive = tk.Label(self.input\_frame\_receive, text="Ввод второго порта:")

82. self.input\_label\_receive.pack(side=tk.LEFT, padx=5)

83. self.input\_entry\_receive = tk.Entry(self.input\_frame\_receive, width=40)

84. self.input\_entry\_receive.pack(side=tk.LEFT, padx=5)

85. self.input\_entry\_receive.bind("<Return>", self.send\_message\_second)

86. self.status\_frame = tk.Frame(self.master)

87. self.status\_frame.pack(pady=10)

88. self.port\_speed\_label = tk.Label(self.status\_frame, text="Скорость порта: 0")

89. self.port\_speed\_label.pack(pady=5)

90. self.bytes\_sent\_label = tk.Label(self.status\_frame, text="Передано байт: 0")

91. self.bytes\_sent\_label.pack(pady=5)

92. self.current\_frame\_label = tk.Label(self.status\_frame, text="Текущий кадр: ")

93. self.current\_frame\_label.pack(pady=5)

94. self.bytes\_sent = 0

95. def update\_status\_window(self, port\_speed, bytes\_sent):

96. self.port\_speed\_label.config(text=f"Скорость порта: {port\_speed}")

97. self.bytes\_sent\_label.config(text=f"Передано байт: {bytes\_sent}")

98. def populate\_ports(self):

99. try:

100. ports = self.logic.populate\_ports()

101. self.send\_port\_combobox['values'] = ports

102. self.read\_port\_combobox['values'] = ports

103. if ports:

104. self.send\_port\_combobox.current(0)

105. self.read\_port\_combobox.current(1)

106. self.update\_connected\_send\_port()

107. self.update\_connected\_read\_port()

108. except ValueError as e:

109. messagebox.showerror("Ошибка", str(e))

110. def update\_connected\_send\_port(self, event=None):

111. selected\_port = self.send\_port\_combobox.get()

112. connected\_port = self.get\_connected\_port(selected\_port)

113. self.send\_connected\_port.config(text=connected\_port)

114. self.update\_read\_ports()

115. def update\_connected\_read\_port(self, event=None):

116. selected\_port = self.read\_port\_combobox.get()

117. connected\_port = self.get\_connected\_port(selected\_port)

118. self.read\_connected\_port.config(text=connected\_port)

119. def update\_read\_ports(self):

120. selected\_send\_port = self.send\_port\_combobox.get()

121. connected\_send\_port = self.get\_connected\_port(selected\_send\_port)

122. available\_ports = [

123. port for port in self.logic.ports\_list

124. if port != selected\_send\_port and port != connected\_send\_port

125. ]

126. self.read\_port\_combobox['values'] = available\_ports

127. if available\_ports:

128. self.read\_port\_combobox.current(0)

129. else:

130. self.read\_port\_combobox.set('')

131. def send\_message(self, input\_entry, output\_text, port\_combobox, is\_first=True):

132. message = input\_entry.get().strip()

133. print(message)

134. if not message:

135. return

136. selected\_port = port\_combobox.get()

137. connected\_port = self.get\_connected\_port(selected\_port)

138. if connected\_port == "Нет связанного порта":

139. messagebox.showerror("Ошибка", "Нет связанного порта для отправки.")

140. return

141. source\_address = str(selected\_port.replace('COM', ''))

142. binary\_message = ''.join(format(ord(char), '08b') for char in message)

143. desired\_length = self.DATA\_SIZE \* 8

144. if len(binary\_message) > desired\_length:

145. binary\_message = binary\_message[:desired\_length]

146. binary\_message = binary\_message.ljust(desired\_length, '0')

147. cyclic\_code = SystematicCyclicCode(len(binary\_message))

148. coded\_message = cyclic\_code.compute\_crc(binary\_message)

149. flag = self.flag[:self.FLAG\_SIZE]

150. destination\_address = self.destination.decode()

151. source\_address = source\_address

152. final\_message\_str = f"{flag}{destination\_address}{source\_address}{coded\_message}"

153. print(f"w-> {final\_message\_str.encode()}")

154. packet = final\_message\_str.encode()

155. parity = self.parity\_combobox\_send.get() if is\_first else self.parity\_combobox\_receive.get()

156. \_

157. \_

158. \_

159. \_

160. \_

161. \_

162. \_

163. \_

164. \_

165. \_

166. \_

167. \_

168. \_

169. \_

170. \_

171. \_

172. \_

173. \_

174. \_

175. \_

176. \_

177. \_

178. \_

179. \_

180. \_

181. \_

182. \_

183. \_

184. \_

185. \_

186. self.update\_output(cyclic\_code, output\_text, response, connected\_port)

187. self.bytes\_sent = self.FLAG\_SIZE + 1 + 1 + self.DATA\_SIZE + cyclic\_code.fcs\_size

188. self.update\_status\_window(self.logic.get\_port\_speed(selected\_port), self.bytes\_sent)

189. def update\_output(self, cyclic\_code, output\_text, message, connected\_port):

190. if message is None:

191. return

192. data\_fcs = message[4:]

193. print(f"data\_fcs : {data\_fcs}")

194. binary\_string = data\_fcs.decode('ascii')

195. print(f"Бинарная строка: {binary\_string}")

196. received\_message = cyclic\_code.introduce\_random\_error(binary\_string)

197. print(f"Полученное сообщение с ошибкой: {received\_message}")

198. corrected\_message = cyclic\_code.check\_and\_correct(received\_message)

199. print(f"Исправленное сообщение: {corrected\_message}")

200. highlighted\_message = cyclic\_code.corrupted\_message

201. print(f"Сообщение с выделенной ошибкой: {highlighted\_message}")

202. self.current\_frame\_label.config(text=f"Принято: {message[:4]}{highlighted\_message}")

203. def binary\_to\_string(binary\_message):

204. chars = [chr(int(binary\_message[i:i + 8], 2)) for i in range(0, len(binary\_message), 8)]

205. return ''.join(chars)

206. corrected\_message\_text = binary\_to\_string(corrected\_message)

207. print(f"\nИсправленное сообщение (текст): {corrected\_message\_text}")

208. output\_text.config(state='normal')

209. output\_text.insert(tk.END, f"w-> {message}\n")

210. output\_text.insert(tk.END, f"r-> {message[:4]}{highlighted\_message}\n")

211. output\_text.insert(tk.END, f"Data -> {corrected\_message\_text}\n")

212. output\_text.config(state='disabled')

213. def send\_message\_first(self, event=None):

214. self.send\_message(self.input\_entry\_send, self.left\_output\_text, self.send\_port\_combobox, is\_first=True)

215. def send\_message\_second(self, event=None):

216. self.send\_message(self.input\_entry\_receive, self.right\_output\_text, self.read\_port\_combobox, is\_first=False)

217. def get\_connected\_port(self, port):

218. return self.logic.port\_pairs.get(port, "Нет связанного порта")

219. def on\_closing(self):

220. self.master.destroy()  
  
Файл SystematicCyclicCode

1. import math

2. import random

3. class SystematicCyclicCode:

4. table\_data = {

5. 1: {'Cтепень полинома': 'х^1 + 1', 'Двоич. код полинома': '11', 'Десятич. код полинома': 3},

6. 2: {'Cтепень полинома': 'х^2 + х + 1', 'Двоич. код полинома': '111', 'Десятич. код полинома': 7},

7. 3: {'Cтепень полинома': 'х^3 + х + 1', 'Двоич. код полинома': '1011', 'Десятич. код полинома': 11},

8. 4: {'Cтепень полинома': 'х^4 + х^3 + х^2 + x + 1', 'Двоич. код полинома': '111111', 'Десятич. код полинома': 31},

9. 5: {'Cтепень полинома': 'х^5 + х^3 + 1', 'Двоич. код полинома': '101001', 'Десятич. код полинома': 41},

10. }

11. def \_\_init\_\_(self, data\_size):

12. self.data\_size = data\_size

13. print(f"Инициализация с размером данных: {self.data\_size}")

14. self.polynomial = self.calculate\_polynomial()

15. self.fcs\_size = self.crc\_bytes\_length(self.polynomial)

16. print(f"Полином: {self.polynomial}, Размер FCS: {self.fcs\_size}")

17. self.check\_bits = None

18. self.corrupted\_message = ""

19. def get\_binary\_code\_polynomial(self, row\_num):

20. entry = self.table\_data.get(row\_num)

21. if entry:

22. print(f"Используем полином для строки {row\_num}: {entry['Двоич. код полинома']}")

23. return entry['Двоич. код полинома']

24. else:

25. print("Используем полином CRC-32 по умолчанию.")

26. return '10011000101110111110101100001111'

27. def calculate\_polynomial(self):

28. self.X\_k = math.ceil(math.log2(self.data\_size + math.ceil(math.log2(self.data\_size + 1))))

29. print(f"Степень полинома (X\_k): {self.X\_k}")

30. return self.get\_binary\_code\_polynomial(self.X\_k)

31. def crc\_bytes\_length(self, polynomial\_bitstring):

32. polynomial\_length = len(polynomial\_bitstring)

33. crc\_length\_bytes = (polynomial\_length - 1 + 7) // 8

34. print(f"Длина полинома: {polynomial\_length}, Длина CRC в байтах: {crc\_length\_bytes}")

35. return crc\_length\_bytes

36. def compute\_crc(self, message):

37. padded\_message = message + '0' \* (len(self.polynomial) - 1)

38. remainder = self.division(padded\_message, self.polynomial)

39. self.check\_bits = remainder

40. return message + remainder

41. def division(self, dividend, divisor):

42. dividend = list(dividend)

43. divisor = list(divisor)

44. divisor\_length = len(divisor)

45. for i in range(len(dividend) - len(divisor) + 1):

46. if dividend[i] == '1':

47. for j in range(divisor\_length):

48. dividend[i + j] = str(int(dividend[i + j]) ^ int(divisor[j]))

49. remainder = ''.join(dividend[-(divisor\_length - 1):])

50. return remainder

51. def check\_and\_correct(self, received\_message):

52. print(f"Полученное сообщение: {received\_message}")

53. remainder = self.division(received\_message, self.polynomial)

54. if '1' not in remainder:

55. print("Ошибок не найдено.")

56. return received\_message[:-len(self.check\_bits)]

57. self.corrupted\_message = received\_message

58. print("Ошибки найдены, начинаем исправление...")

59. for i in range(len(received\_message)):

60. modified\_message = list(received\_message)

61. modified\_message[i] = '0' if modified\_message[i] == '1' else '1'

62. modified\_message = ''.join(modified\_message)

63. new\_remainder = self.division(modified\_message, self.polynomial)

64. if '1' not in new\_remainder:

65. self.corrupted\_message = ''.join(received\_message[:i]) + f"\_\_{received\_message[i]}\_\_" + ''.join(received\_message[i + 1:])

66. print(f"Исправленное сообщение: {modified\_message}")

67. return modified\_message[:-len(self.check\_bits)]

68. print("Ошибка не может быть исправлена")

69. return "Ошибка не может быть исправлена"

70. def introduce\_random\_error(self, message):

71. message\_list = list(message)

72. info\_bits\_length = len(message) - len(self.check\_bits)

73. if random.random() < 1:

74. index\_to\_corrupt = random.randint(0, info\_bits\_length - 1)

75. original\_bit = message\_list[index\_to\_corrupt]

76. message\_list[index\_to\_corrupt] = '0' if original\_bit == '1' else '1'

77. print(f"\nCorrupt bit at index {index\_to\_corrupt} changed from {original\_bit} to {message\_list[index\_to\_corrupt]}")

78. return ''.join(message\_list)

79. def convert\_to\_byte\_message(self, binary\_message, error\_position=None):

80. byte\_message = bytearray()

81. for i in range(0, len(binary\_message), 8):

82. byte = binary\_message[i:i + 8]

83. if len(byte) == 8:

84. if error\_position is not None and (i <= error\_position < i + 8):

85. byte\_message.append(int(byte, 2))

86. else:

87. byte\_message.append(int(byte, 2))

88. formatted\_message = ''.join(f"\\x{byte:02x}" for byte in byte\_message)

89. print(f"Преобразованное байтовое сообщение: {formatted\_message}")

90. return formatted\_message

91. def convert\_bytes\_to\_binary\_message(self, byte\_string):

92. return ''.join(format(byte, '08b') for byte in byte\_string)