**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.

**1.2 Структура пакета**

Структура пакета (или кадра) является ключевым элементом в процессе передачи данных через последовательные порты. Правильное оформление пакета позволяет обеспечить целостность и корректность передаваемой информации. В рамках лабораторной работы структура пакета будет включать следующие поля:

1. Флаг. Флаг указывает на начало пакета. Он представляет собой двоичное значение, эквивалентное длине поля данных в байтах. Занимает фиксированное количество бит, обычно равное 8 битам (1 байт), что позволяет выразить значения от 0 до 255.

2. Destination Address (Адрес назначения). Поле, указывающее на адрес устройства, для которого предназначен пакет. В данной реализации оно будет установлено в нулевое значение. Занимает 4 бита, что позволяет использовать до 16 различных адресов.

3. Source Address (Адрес источника). Поле, содержащее номер используемого COM-порта, который отправляет данные. Это позволяет получателю понять, откуда пришел пакет. Занимает 4 бита, аналогично полю адреса назначения.

4. Data (Данные). Основное поле, содержащее передаваемую информацию. Длина этого поля фиксирована и равна n байтам, где n — номер группы. Например, если n равно 4, поле данных будет занимать 32 бита (4 байта).

5. FCS (Frame Check Sequence). Поле контрольной суммы, используемое для проверки целостности пакета. В данной реализации оно будет установлено в нулевое значение. Занимает 1 байт (8 бит), что позволяет обеспечить базовую проверку данных.

Структура пакета, описанная выше, обеспечивает необходимую организацию данных для передачи по COM-портам. Определение фиксированных размеров полей помогает гарантировать правильное чтение и обработку пакетов как на стороне отправителя, так и на стороне получателя.

**1.3 Алгоритм бит-стаффинга на стороне передатчика**

Алгоритм бит-стаффинга используется для предотвращения потери синхронизации в передаче данных. Ниже представлена блок-схема алгоритма на стороне передатчика, описывающая процесс бит-стаффинга.

**1.4 Алгоритм де-бит-стаффинга на стороне приемника**

Алгоритм де-бит-стаффинга используется для восстановления исходной последовательности битов, удаляя добавленные биты, которые были вставлены для предотвращения длинных последовательностей единиц. Ниже представлена блок-схема алгоритма 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.

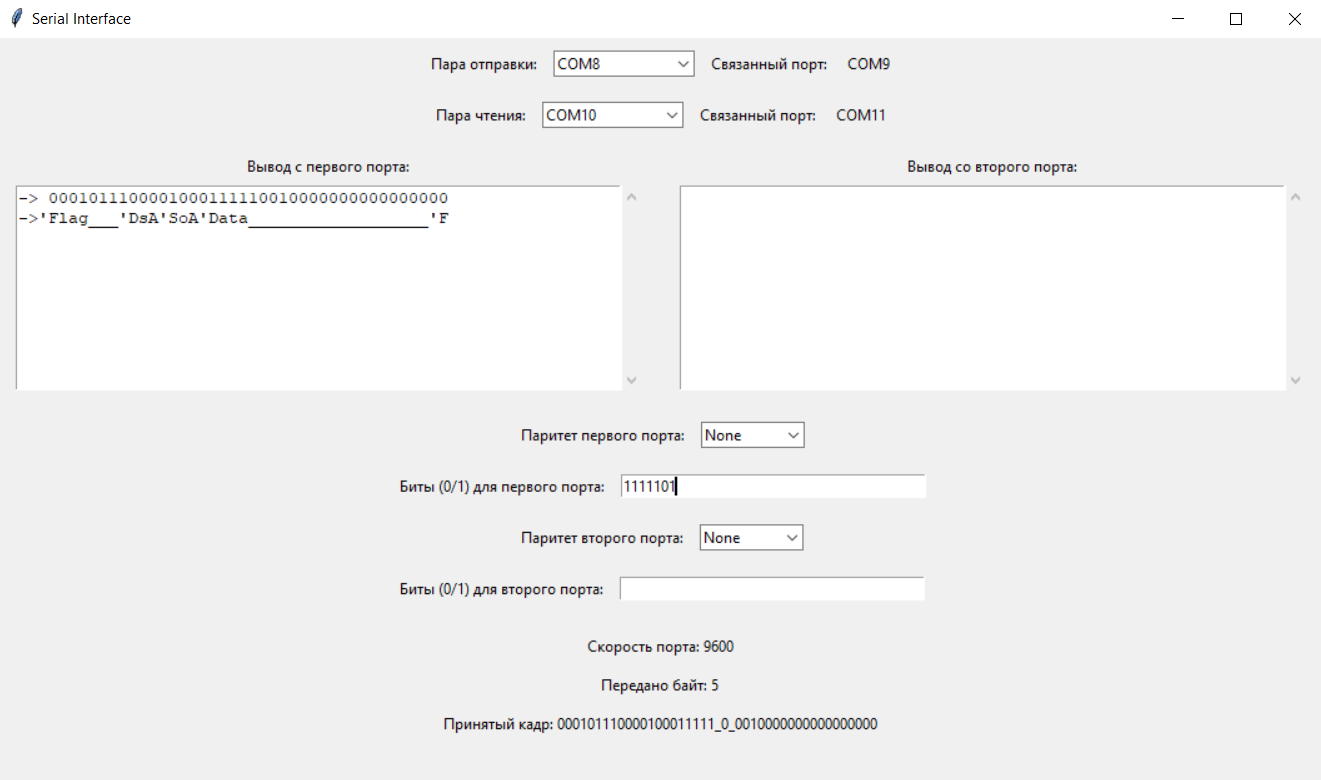


Рисунок 2.1 – Интерфейс программы

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. class SerialLogic:

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

7. self.port\_pairs = {}

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

9. self.read\_threads = []

10. self.port\_speeds = {}

11. self.baudrate = 9600

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

13. return self.baudrate

14. def populate\_ports(self):

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

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

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

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

19. self.create\_port\_pairs()

20. return self.ports\_list

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

22. checked\_ports = set()

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

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

25. if port1 in checked\_ports:

26. continue

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

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

29. if port2 in checked\_ports:

30. continue

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

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

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

34. checked\_ports.add(port1)

35. checked\_ports.add(port2)

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

37. break

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

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

40. try:

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

42. test\_message = "TEST"

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

44. if ser2.in\_waiting > 0:

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

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

47. return response == test\_message

48. else:

49. print("Нет ответа от порта.")

50. return False

51. except serial.SerialException as e:

52. print(f"Ошибка: {e}")

53. return False

54.

55.

56.

57.

58.

59.

60.

61.

62.

63.

64.

65.

66.

67.

68.

69.

70.

71.

72.

73.

74.

75.

76.

77.

78.

79.

80.

81.

82.

83.

84.

85.

86.

87.

88.

89.

90.

91.

92.

93.

94.

95.

96.

97.

98.

99.

100.

101.

102.

103.

104.

105.

106.

107.

108.

109.

110.

111.

112.

113.

114.

115.

116.

117.

Файл serial\_interface.py

1. import tkinter as tk

2. from tkinter import scrolledtext, ttk, messagebox

3. from serial\_logic import SerialLogic

4. class SerialInterfaceApp:

5.

6.

7.

8.

9.

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

11. self.master = master

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

13. self.logic = SerialLogic()

14. self.master.minsize(800, 600)

15. self.setup\_ui()

16. self.populate\_ports()

17. def setup\_ui(self):

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

41. self.output\_frame.pack(pady=10)

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

43. self.left\_output\_frame.pack(side=tk.LEFT, padx=10)

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

45. self.right\_output\_frame.pack(side=tk.RIGHT, padx=10)

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

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

48. self.left\_output\_text = scrolledtext.ScrolledText(self.left\_output\_frame, width=60, height=10, state='disabled')

49. self.left\_output\_text.pack(padx=5, pady=5)

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

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

52. self.right\_output\_text = scrolledtext.ScrolledText(self.right\_output\_frame, width=60, height=10, state='disabled')

53. self.right\_output\_text.pack(padx=5, pady=5)

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

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

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

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

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

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

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

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

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

63. self.input\_label\_send = tk.Label(self.input\_frame\_send, text="Биты (0/1) для первого порта:")

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

65. vcmd\_send = (self.master.register(self.validate\_input), '%S')

66. self.input\_entry\_send = tk.Entry(self.input\_frame\_send, width=40, validate='key', validatecommand=vcmd\_send)

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

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

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

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

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

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

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

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

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

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

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

78. self.input\_label\_receive = tk.Label(self.input\_frame\_receive, text="Биты (0/1) для второго порта:")

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

80. vcmd\_receive = (self.master.register(self.validate\_input), '%S')

81. self.input\_entry\_receive = tk.Entry(self.input\_frame\_receive, width=40, validate='key', validatecommand=vcmd\_receive)

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

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

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

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

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

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

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

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

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

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

92. self.bytes\_sent = 0

93.

94.

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()

133. if not message:

134. return

135. if not all(bit in '01' for bit in message):

136. messagebox.showerror("Ошибка", "Введите только символы '0' и '1'.")

137. return

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

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

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

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

142. return

143. flag = format(self.DATA\_SIZE, '0{}b'.format(self.FLAG\_SIZE))

144. destination\_address = '0000'

145. source\_address = format(int(selected\_port.replace('COM', '')), '0{}b'.format(self.SOURCE\_ADDRESS\_SIZE))

146. if len(message) > self.DATA\_SIZE:

147. message = message[:self.DATA\_SIZE]

148. stuffed\_data = self.logic.bit\_stuffing(message)

149. data = stuffed\_data.ljust(self.DATA\_SIZE, '0')

150. fcs = '0' \* self.FCS\_SIZE

151. packet = f"{flag}{destination\_address}{source\_address}{data}{fcs}"

152. expected\_length = (self.FLAG\_SIZE + self.DESTINATION\_ADDRESS\_SIZE + self.SOURCE\_ADDRESS\_SIZE + len(data) + self.FCS\_SIZE)

153. if len(packet) != expected\_length:

154. messagebox.showerror("Ошибка", "Некорректная длина пакета.")

155. return

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

157. response = self.logic.send\_packet(connected\_port, selected\_port, packet, parity)

158. self.update\_output(output\_text, response, connected\_port)

159. self.bytes\_sent += len(packet) // 8

160. self.update\_status\_window(self.logic

.get\_port\_speed(selected\_port), self.bytes\_sent)

161. def update\_output(self, output\_text, message, connected\_port):

162. highlighted\_message = self.logic.highlight\_bit\_stuffing(message)

163. self.current\_frame\_label.config(text=f"Принятый кадр: {highlighted\_message}")

164. unstuffed\_message = self.logic.de\_bit\_stuffing(message)

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

166. output\_text.insert(tk.END, f"-> {unstuffed\_message}\n")

167. output\_text.insert(tk.END, f"->'Flag\_\_\_'DsA'SoA'Data\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_'F \n")

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

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

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

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

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

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

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

175. def on\_closing(self):

176. self.master.destroy()