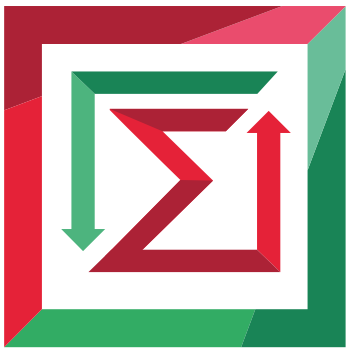
Министерство науки и высшего образования
Российской Федерации

Федеральное государственное бюджетное образовательное
учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

НГТУ
НЭТИ

Кафедра теоретической и прикладной информатики

Лабораторная работа № 1
по дисциплине «Проектирование Систем Реального Времени»
Работа с последовательным портом персонального компьютера



ФАКУЛЬТЕТ:	ПМИ
ГРУППА:	ПМИМ-01
СТУДЕНТЫ:	Ершов П. К. Грициенко И. Г.
БРИГАДА:	7
ПРЕПОДАВАТЕЛЬ:	Кобылянский В. Г.

Новосибирск
2021

1. Цель работы

Целью работы является изучение принципов работы и методов программирования COM портов и изучение различных алгоритмов вычисления контрольной суммы при передаче данных через COM порт.

2. Задание на лабораторную

1. Написать и отладить программу на языке С, определяющую базовые адреса последовательных портов. Сравнить результаты ее работы с результатами программы, приведенной в разделе 3.
2. Написать программу для передачи и приема данных через COM-порт на любом языке программирования. Основные требования:
   - возможность конфигурирования порта (скорость передачи, контроль четности, паритет четности);
   - возможность передачи пакета по ASCII – протоколу и по бинарному протоколу в зависимости от выбора пользователя
3. Соединить COM порты компьютера нуль – модемным кабелем. Если в терминальном классе установлены компьютеры с одним COM портом, то соединить два соседних компьютера. Если COM-порты отсутствуют, то используйте виртуальные порты.
4. Запустить 2 копии программы, одну подключить к порту COM1, вторую – к COM2. Провести обмен данными.
5. В окне настройки порта COM1 включить контроль четности в состояние «четное», провести обмен данными, пояснить результаты.
6. В окне настройки порта COM1 отключить контроль четности и включить скорость передачи данных 9600 бод , провести обмен данными, пояснить результаты.
7. Отладить программу и занести в протокол результаты ее работы.
8. Написать и отладить функции вычисления простой контрольной суммы, LRC, CRC16 и CRC32. Добавить эти функции к программе, разработанной в п. 2.

9. Провести тестирование функций для передачи символьной строки согласно Вашего варианта задания (Программа).
10. Изменить один байт в посылке и найти контрольные суммы, результаты занести в протокол.
11. Поменять местами два любых символа в посылке и найти контрольные суммы, результаты занести в протокол.

3. Ход работы.

3.1. Выполнение данного пункта невозможно в силу ограничений операционной системы Windows 10 на чтение памяти.

3.2. Программа успешно разработана. Было принято решение выполнять все тесты в общей программе.

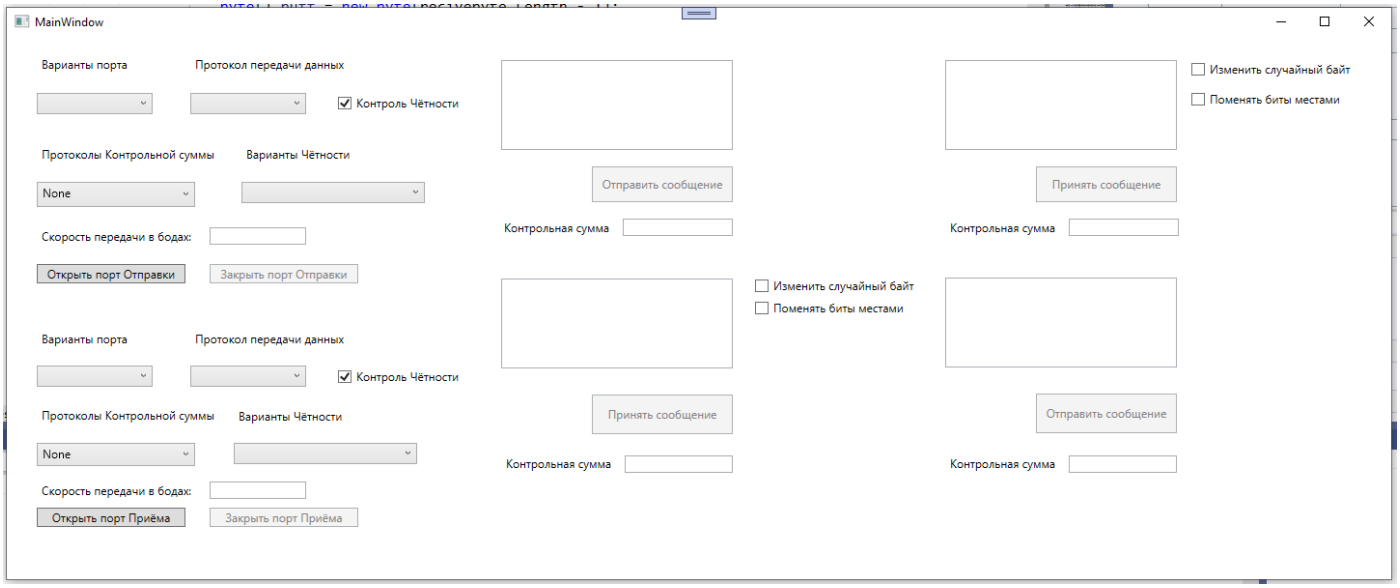


Рисунок 1. Интерфейс программы

3.3. Соединение успешно выполнено с помощью программы Virtual Null Modem

Рисунок 2. Конфигурация виртуального нуль-модемного кабеля

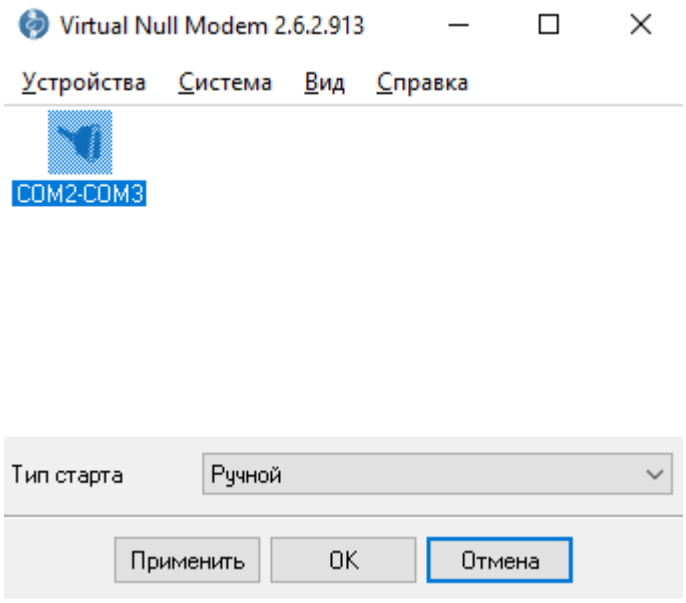


Рисунок 3. Активное виртуально нуль-модемное соединение

3.4. Порты COM2 и COM3 (в силу того, что программа Virtual Null Modem не позволяет подключиться к COM1, были использованы полностью виртуальные последовательные порты) успешно открыты и произведён обмен сообщениями в обе стороны.

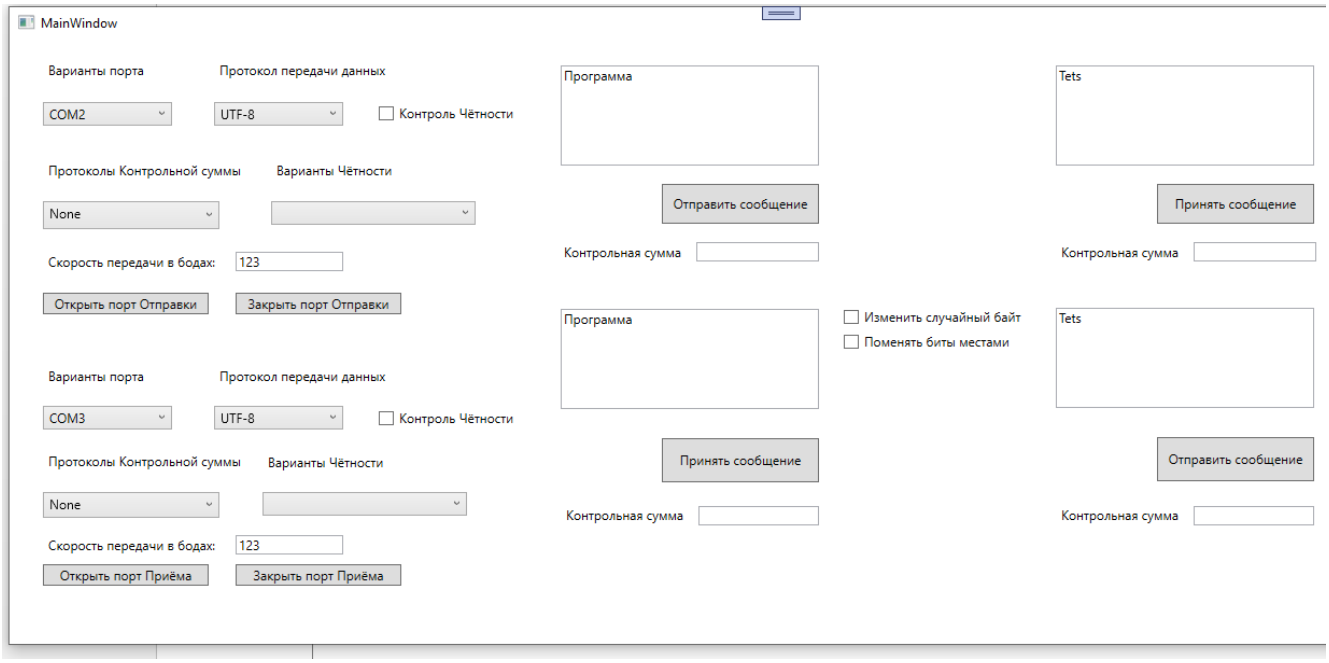


Рисунок 4. Успешный обмен сообщением через виртуальное нуль-модемное соединение

3.5. Попытка считывания сообщения при включённом контроле чётности на порте-отправителе и отключённом контроле чётности на порте-получателе оказалась неудачной. Считывание не произошло.

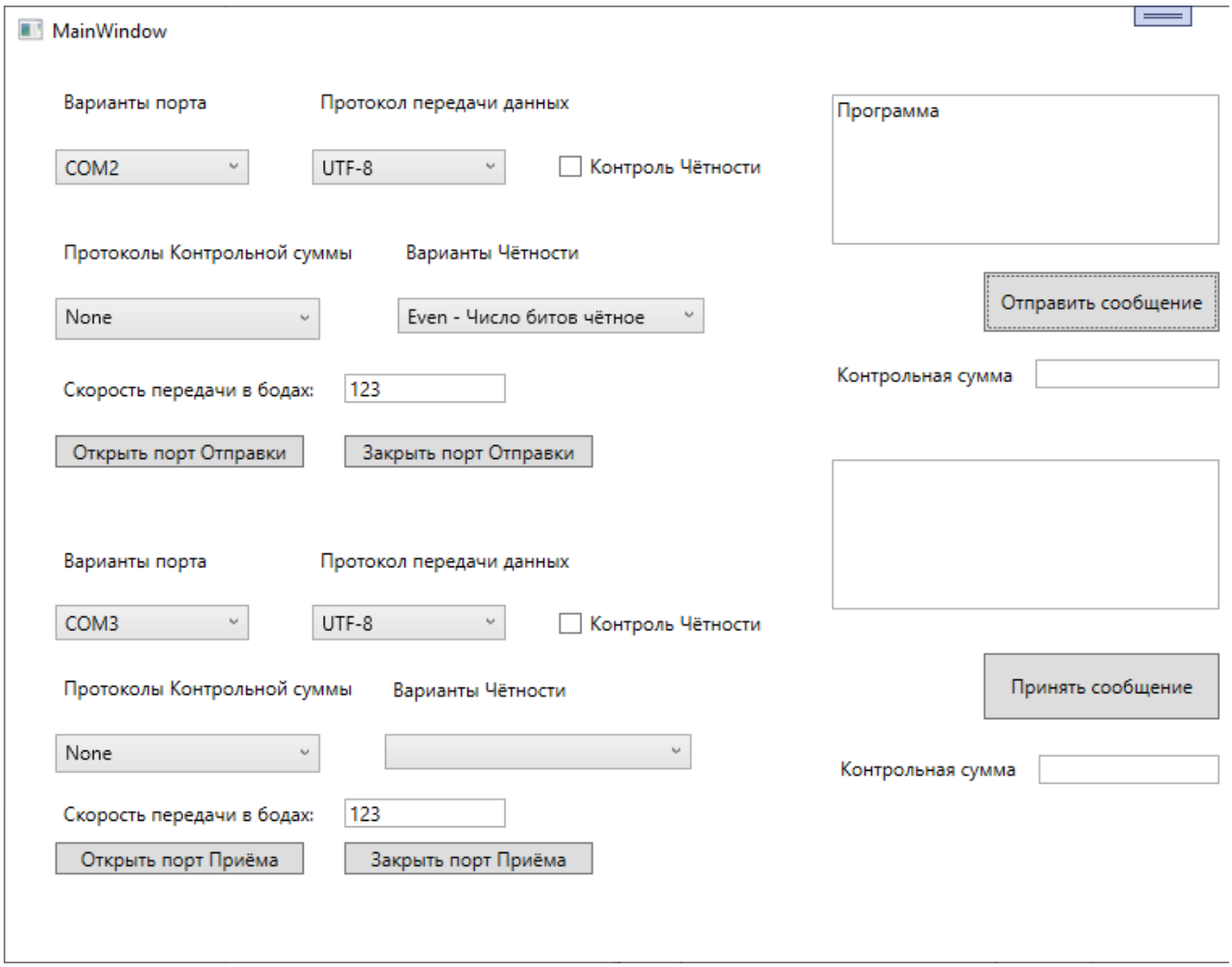


Рисунок 5. Неудачное считывание сообщения

3.6. При отключении контроля чётности удаётся успешно получить сообщение.

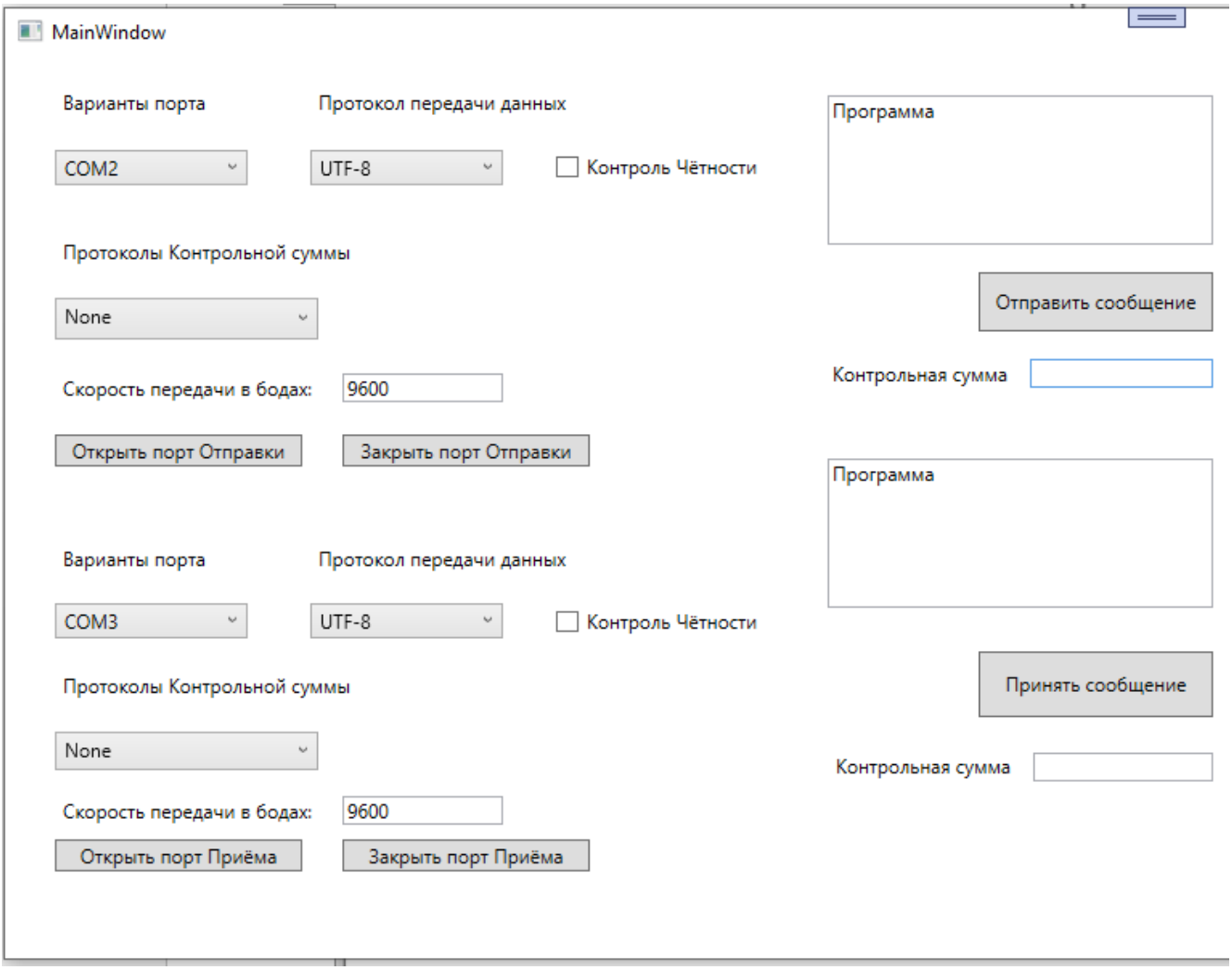


Рисунок 6. Успешно чтение восстановилось

3.7. Провести тестирование методов вычисления контрольной суммы для передачи строки из варианта (будет использоваться английский перевод слова - Program)

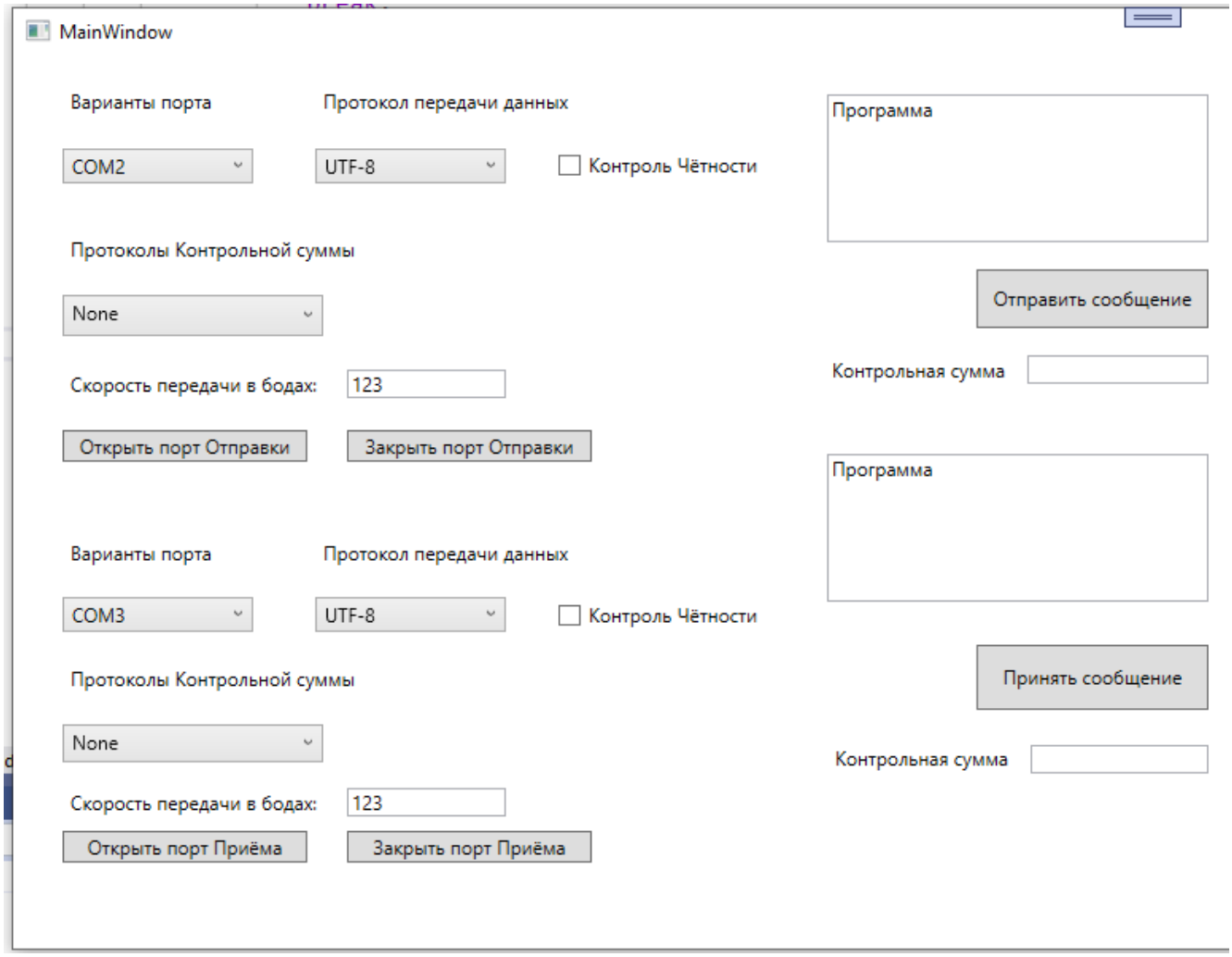


Рисунок 7. Простая передача сообщения

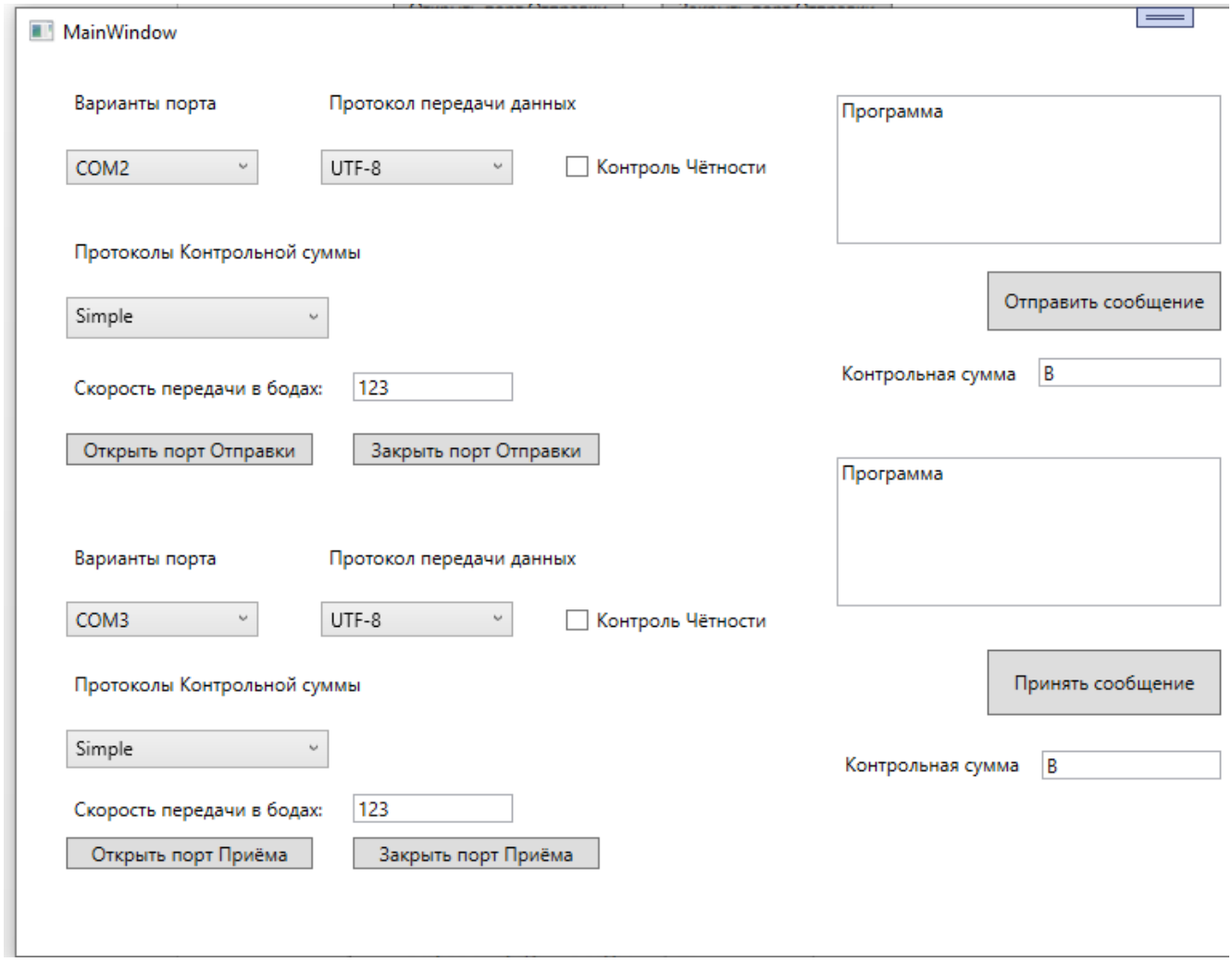


Рисунок 8. Передача с Простой Контрольной суммой успешна

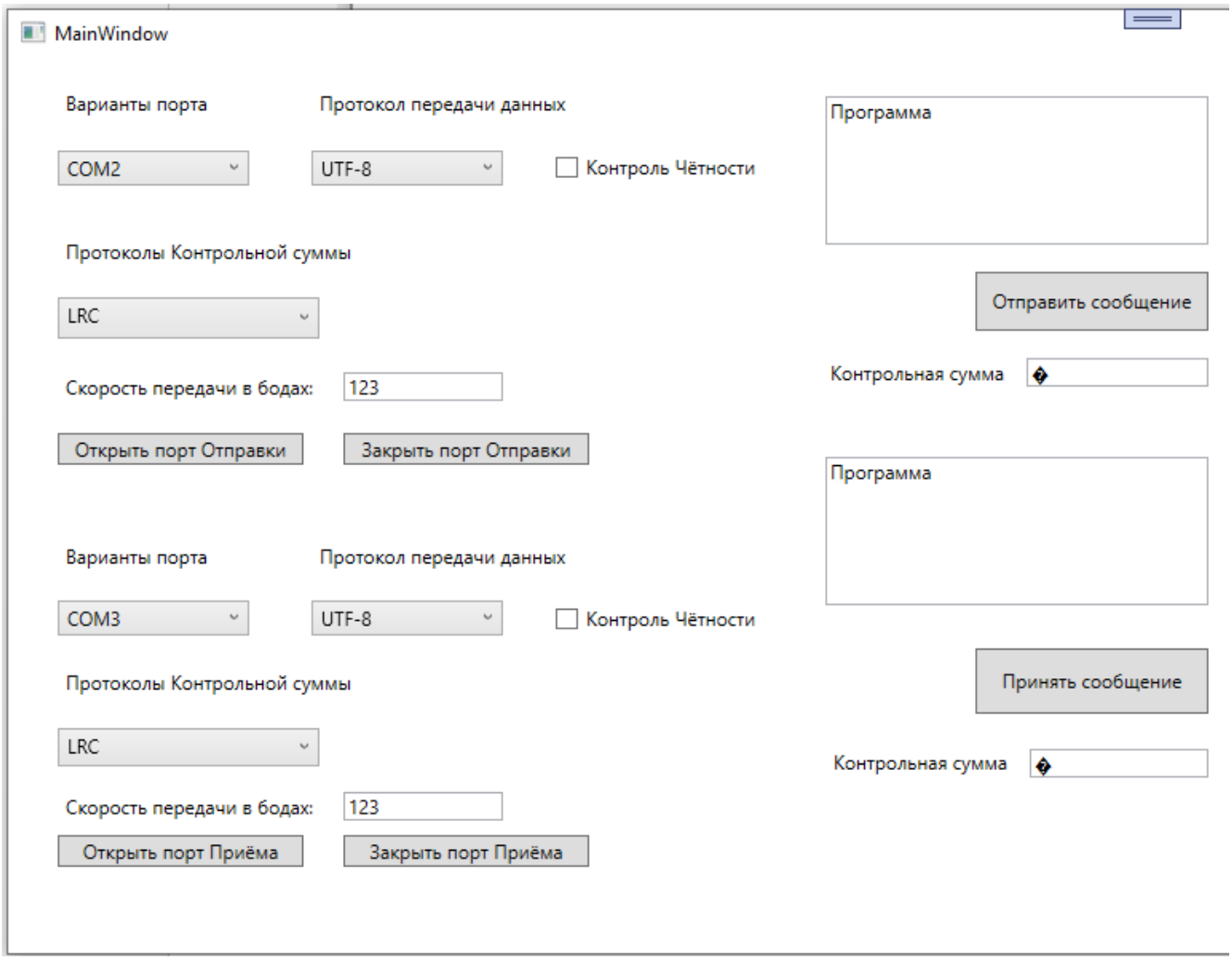


Рисунок 8. Передача с использование LRC успешна

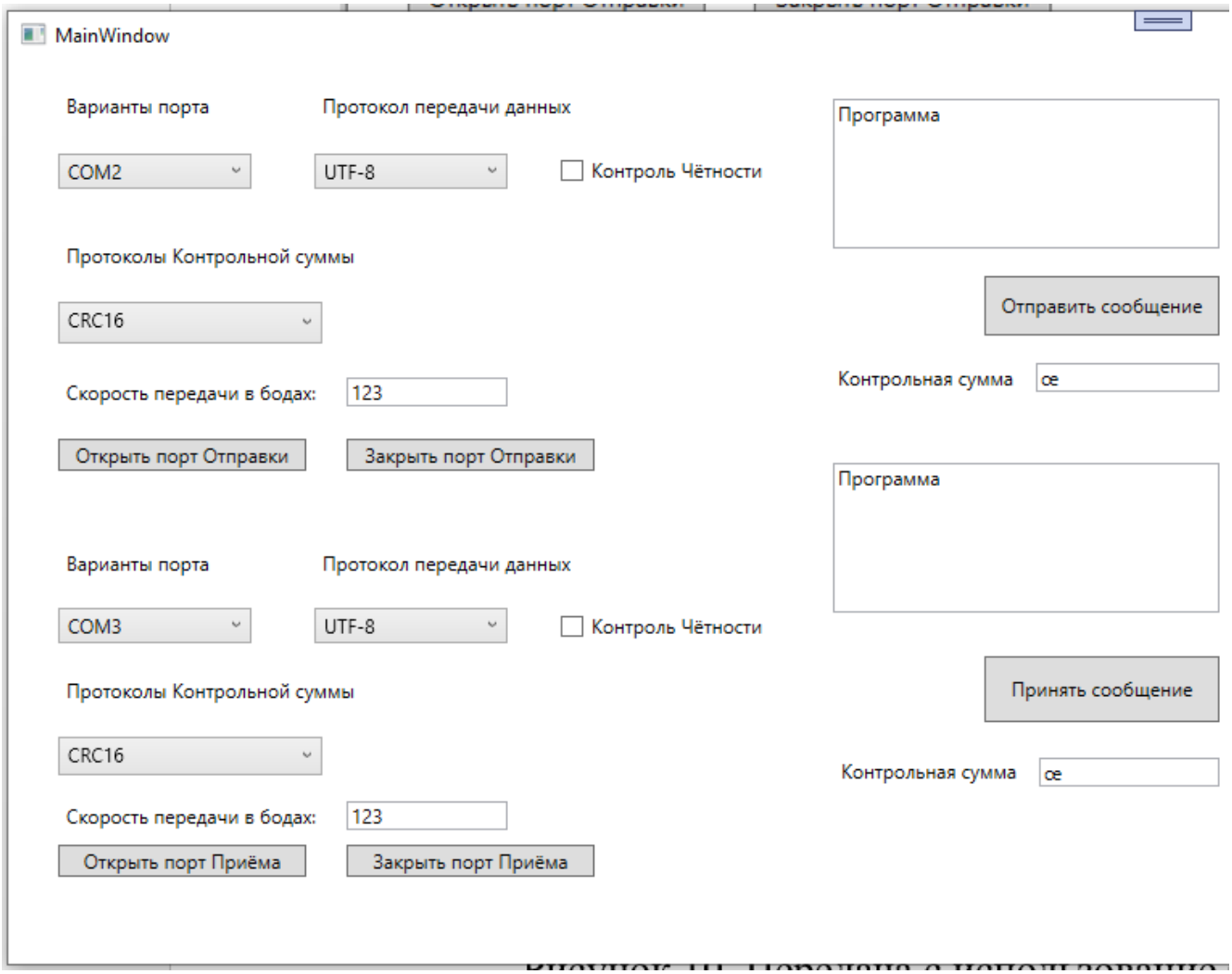


Рисунок 9. Передача с использование CRC16 успешна

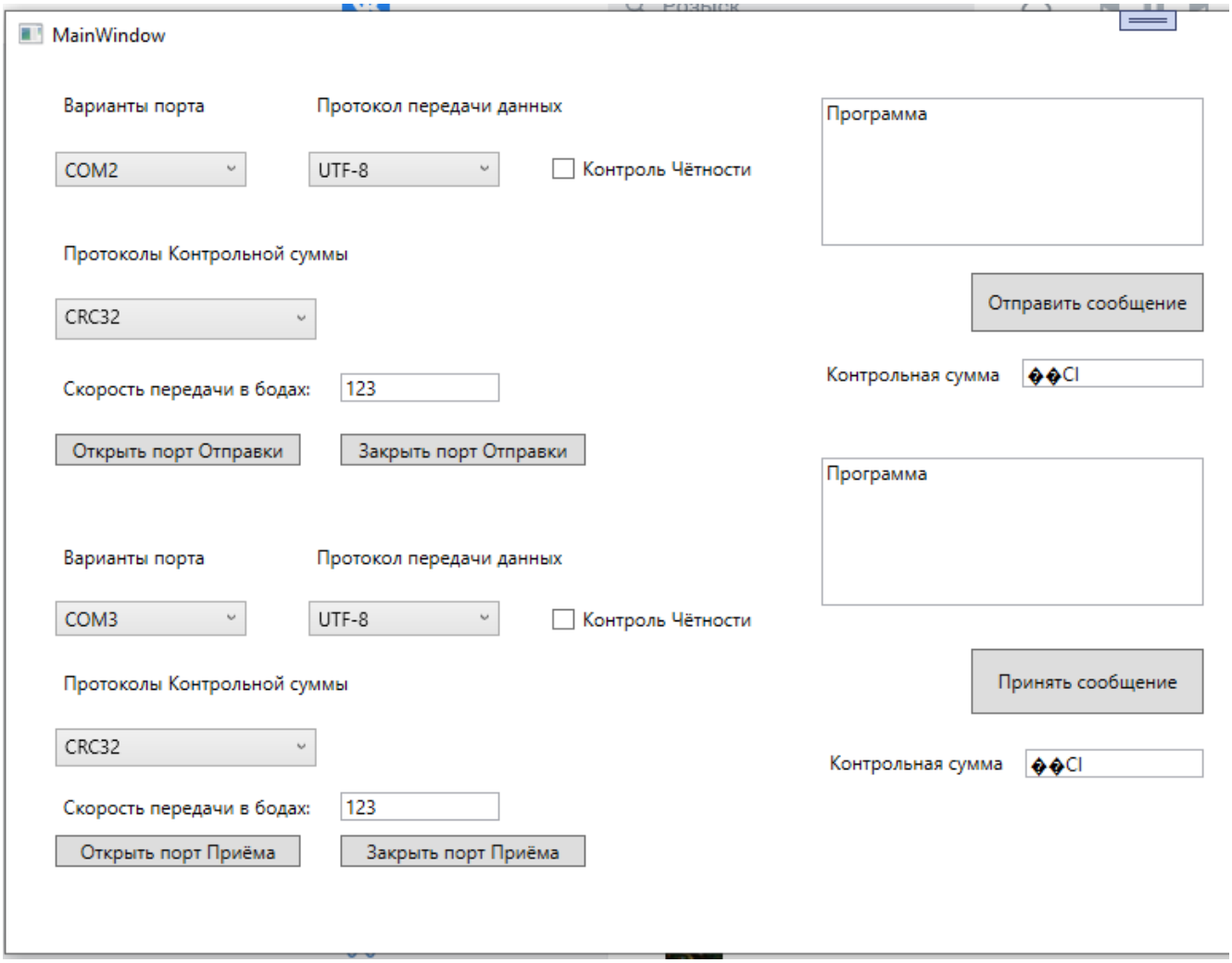


Рисунок 10. Передача с использование CRC32 успешна

3.8. Изменение одного байта с целью проверки работы алгоритмов контрольных сумм.

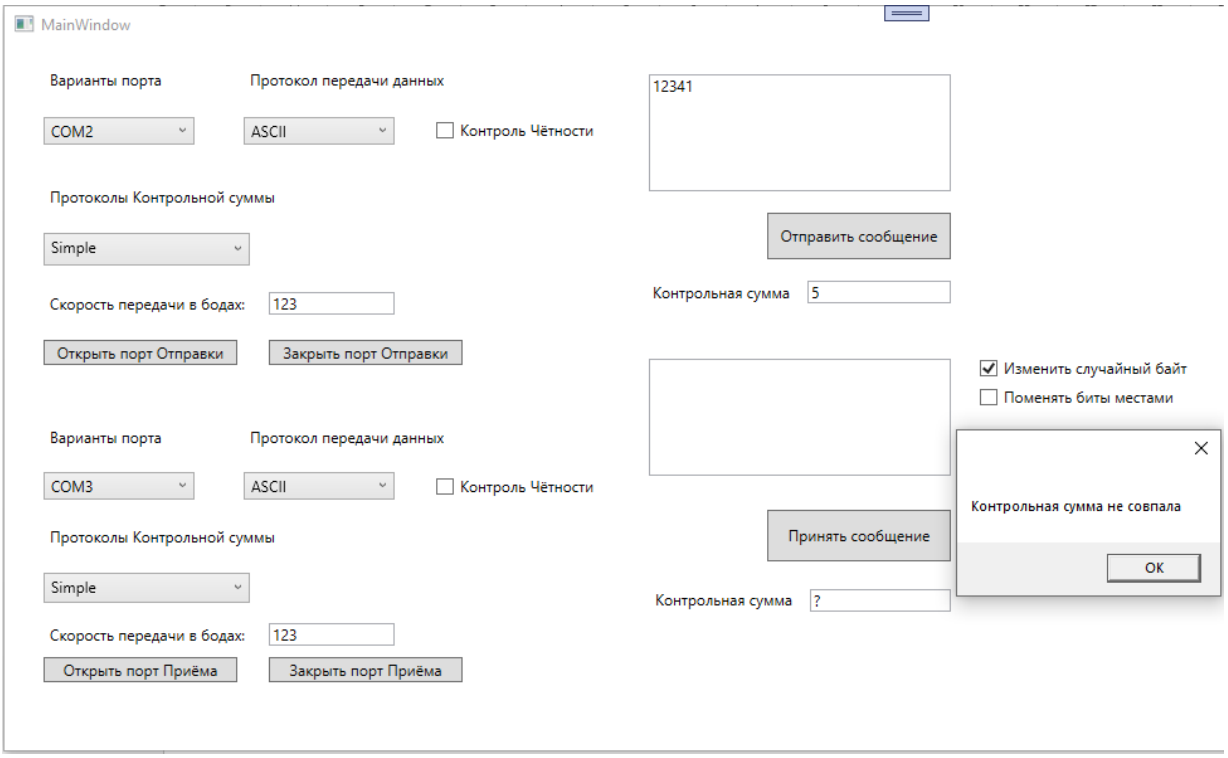


Рисунок 11. Несовпадение контрольных сумм для Простой контрольной суммы

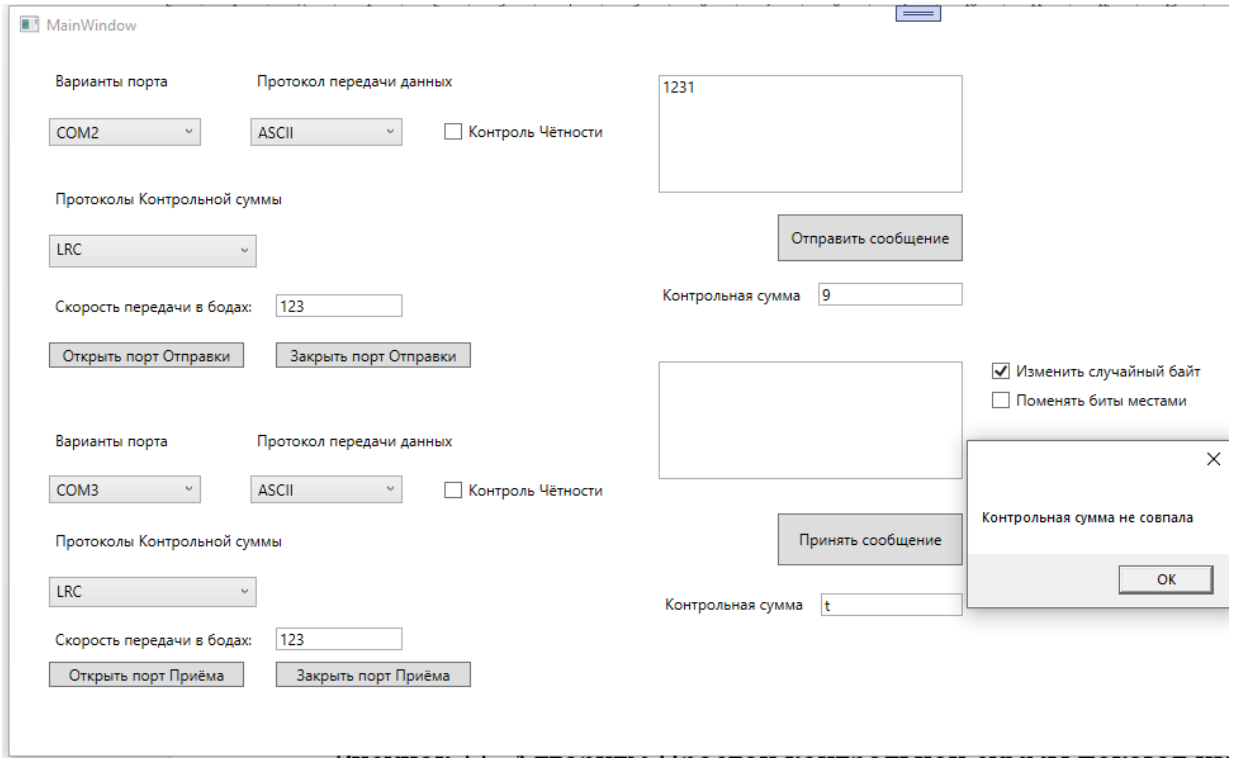


Рисунок 12. Несовпадение контрольных сумм для LRC

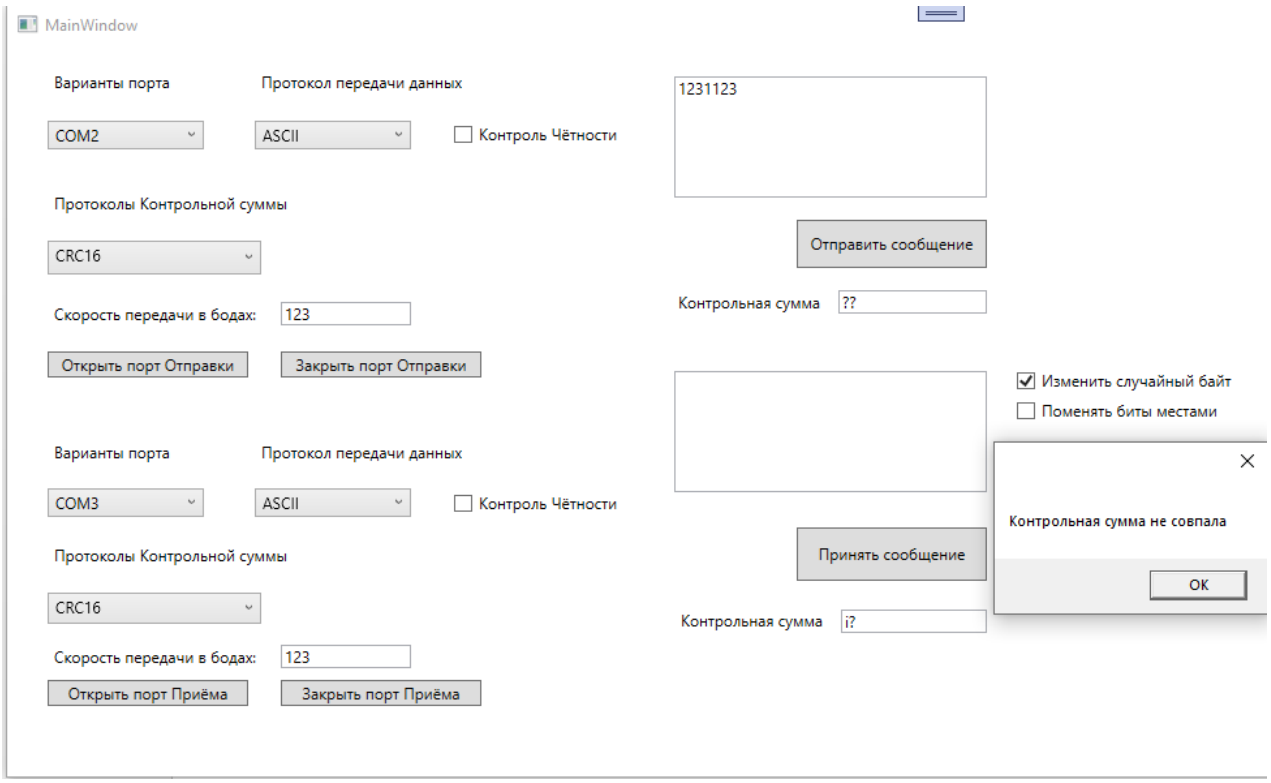


Рисунок 13. Несовпадение контрольных сумм для CRC16

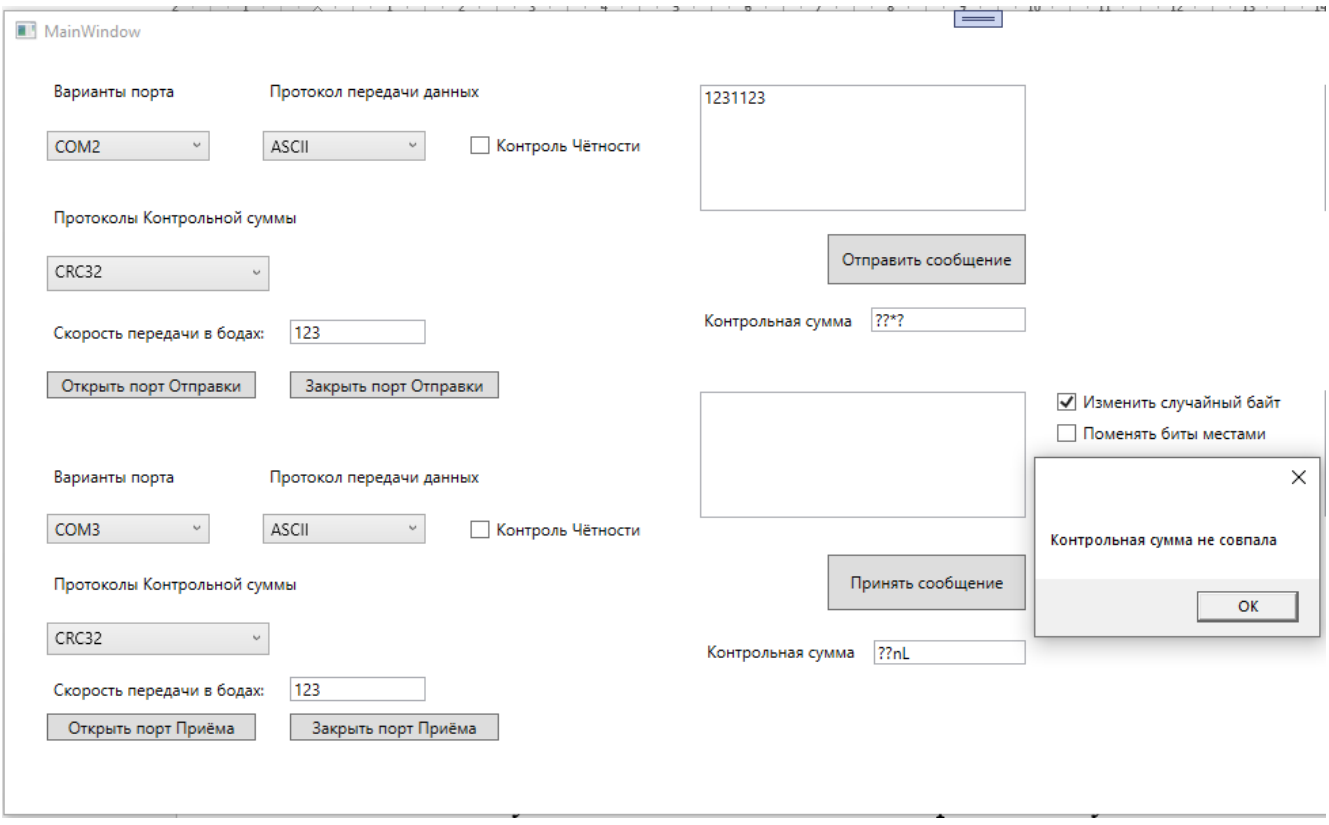


Рисунок 14. Несовпадение контрольных сумм для CRC32

3.9. Два байта меняются местами с целью проверки работы алгоритмов контрольных сумм

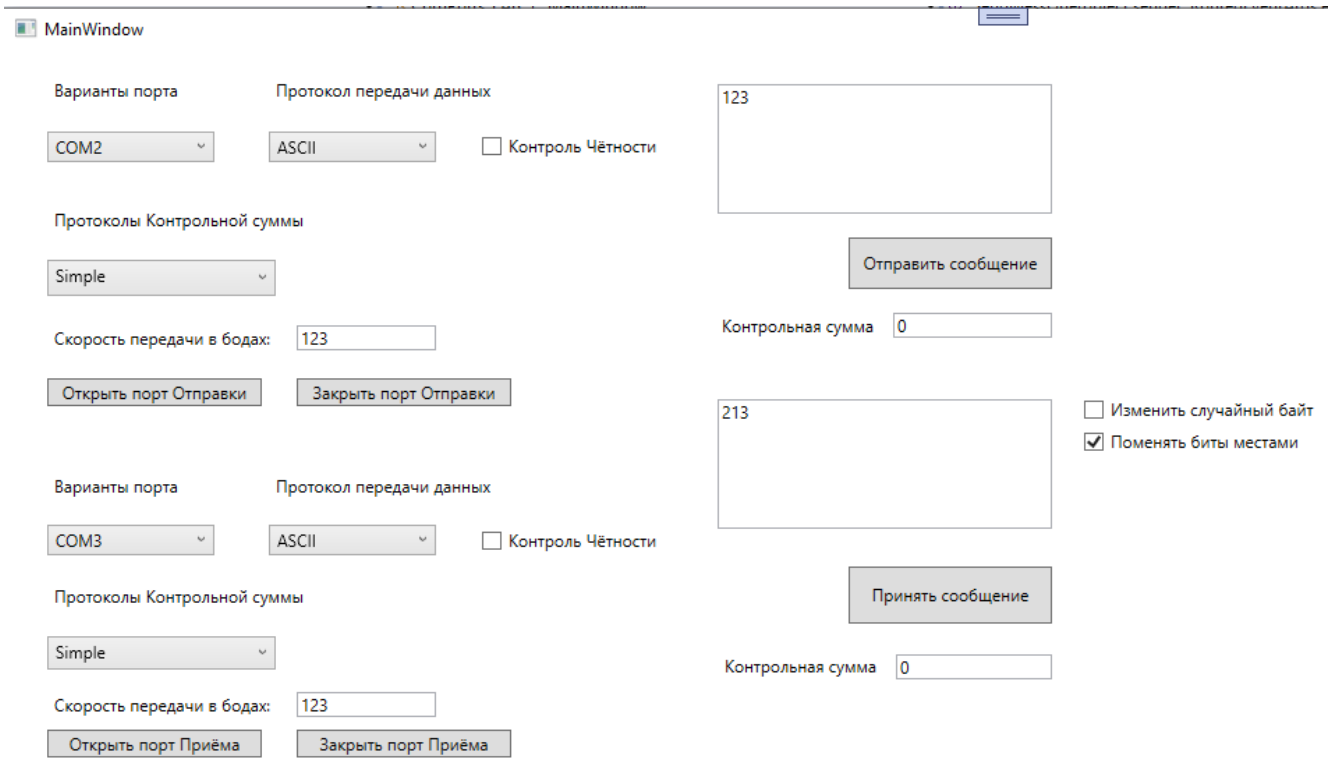


Рисунок 15. Алгоритм простой контрольной суммы не показал разницы между контрольными суммами

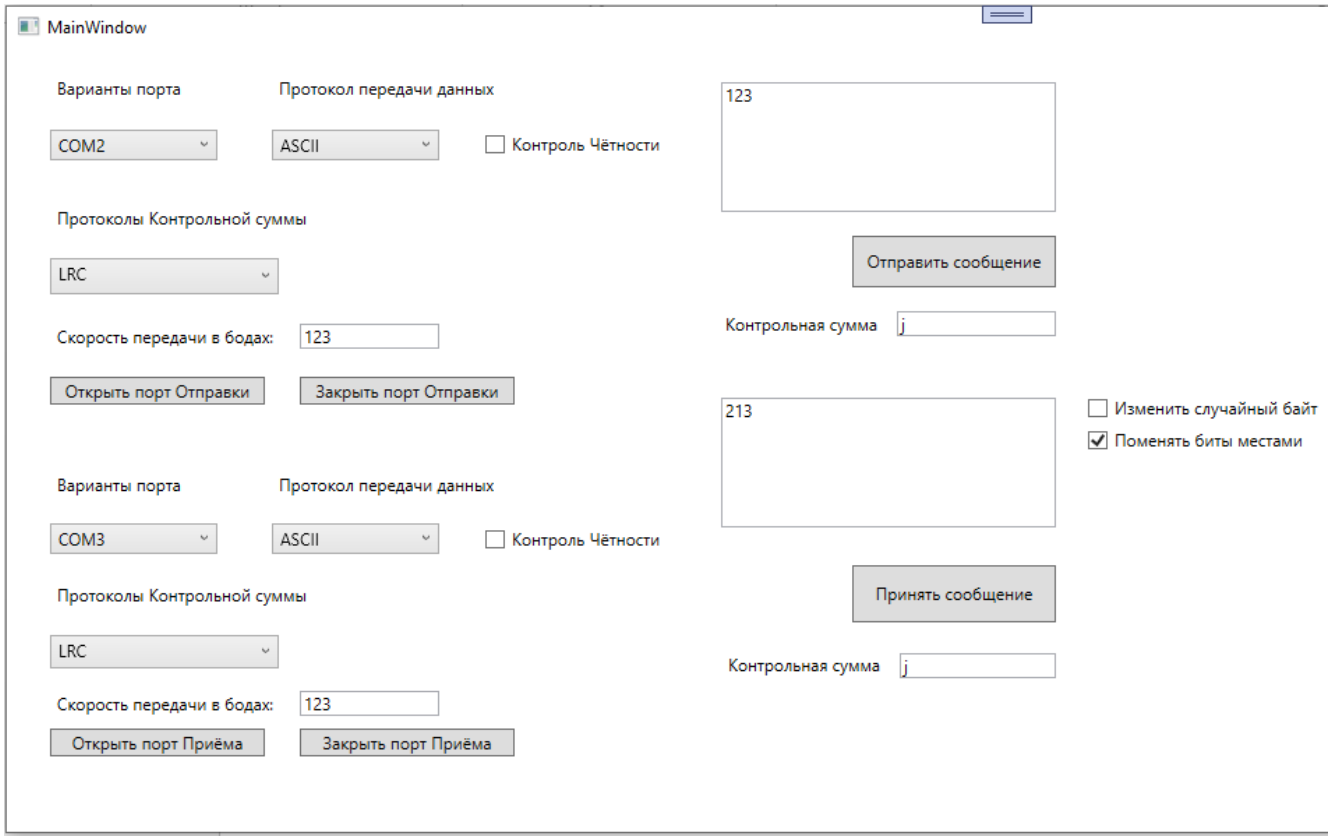


Рисунок 16. Алгоритм LRC не показал разницы между контрольными суммами

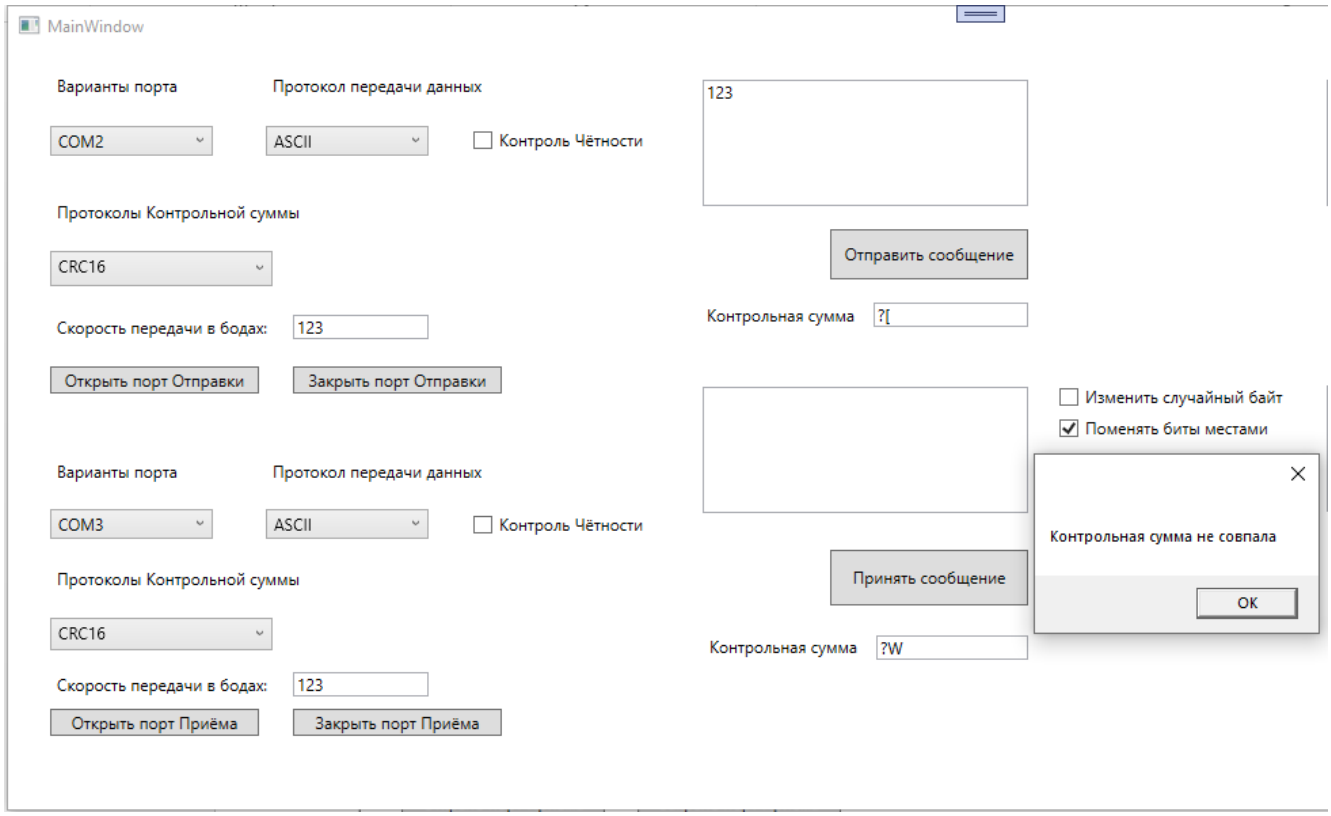


Рисунок 17. Несовпадение контрольных сумм для CRC16

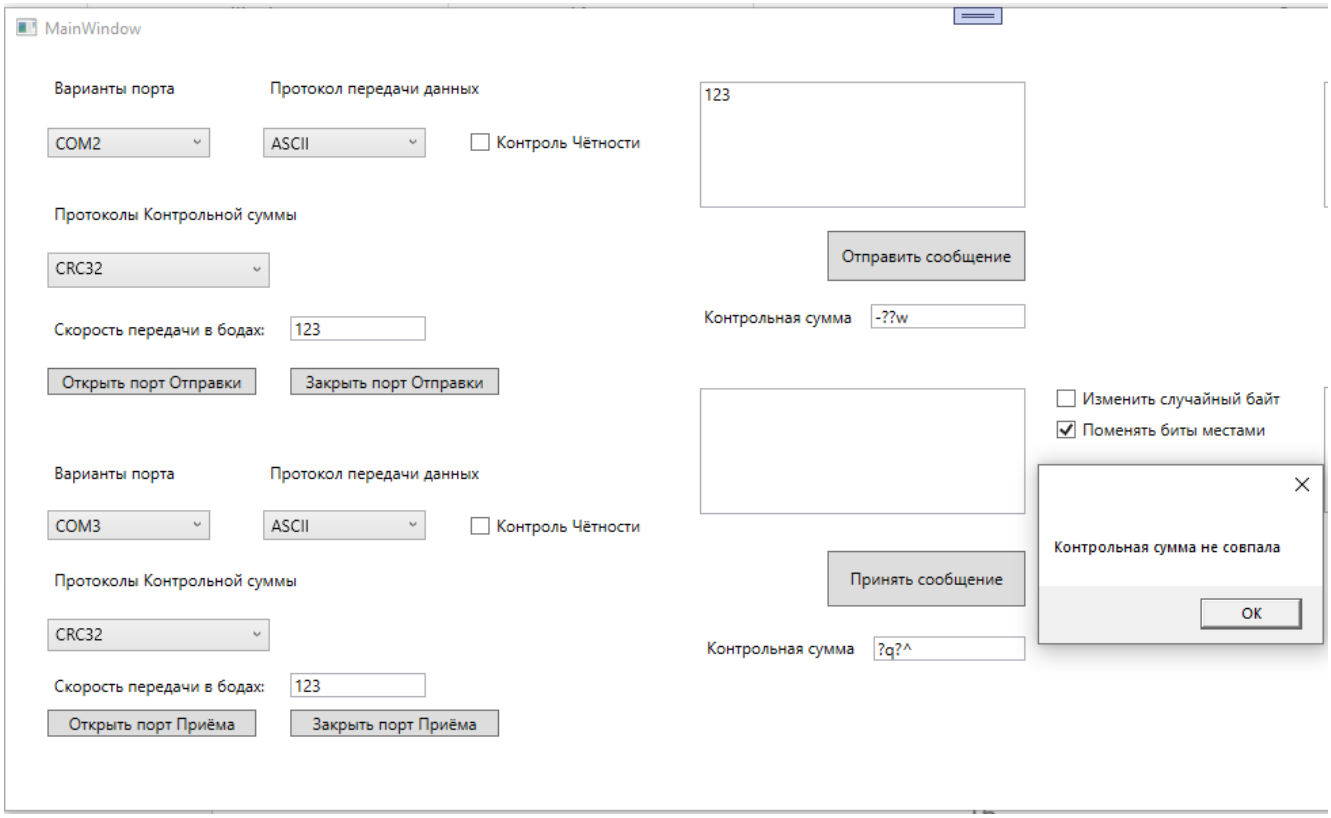


Рисунок 18. Несовпадение контрольных сумм для CRC32

4. Код программы

MainWindow.xaml.cs

```
public partial class MainWindow : Window
    {
        private SerialPort comPort;
        private SerialPort comPort1;
        private bool ModBitOne = false;
        private bool ModBitTwo = false;
        private bool SwapBitOne = false;
        private bool SwapBitTwo = false;

        private string SendPotocol = "";
        private string RecivePotocol = "";

        public string recivetext = "";
        public byte[] recivebyte;
        byte[] sendLRC = new byte[] { };
        byte[] sendSimple = new byte[] { };
        byte[] sendCRC16 = new byte[] { };
        byte[] sendCRC32 = new byte[] { };

        byte[] reciveLRC = new byte[] { };
```

```
        byte[] reciveSimple = new byte[] { };
        byte[] reciveCRC16 = new byte[] { };
        byte[] reciveCRC32 = new byte[] { };
        public MainWindow()
        {

            InitializeComponent();
        }



        private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            List<string> port = new List<string>(COMPortUtil.GetPorts());
            foreach (var i in port.Skip(1))
            {
                COMListRecive.Items.Add(i);
                COMListSend.Items.Add(i);
            }
            ParitySend.Visibility = Visibility.Hidden;
            ParityRecive.Visibility = Visibility.Hidden;
            ParitySendLabel.Visibility = Visibility.Hidden;
            ParityReciveLabel.Visibility = Visibility.Hidden;

            Send.IsEnabled = false;
            Recive.IsEnabled = false;
            SendTwo.IsEnabled = false;
            ReciveTwo.IsEnabled = false;

            CloseSendPort.IsEnabled = false;
            CloseRecivePort.IsEnabled = false;
        }

        public void SendPort(object sender, RoutedEventArgs e)
        {
            if (SendBaud.Text == "")
                MessageBox.Show("Установите скорость передачи данных в бодах");
            else
            {
                if (!(bool)SendParityCheck.IsChecked)
                    comPort = new SerialPort(COMListSend.SelectedItem.ToString(), Convert.ToInt32(SendBaud.Text),
Parity.None, 5, StopBits.One);
                else
                {
```

```
                    switch (ParitySend.Text.Split(" ")[0])
                    {
                        case "Odd":
                            {
                                comPort = new SerialPort(COMListSend.SelectedItem.ToString(),
Convert.ToInt32(SendBaud.Text), Parity.Odd, 5, StopBits.One);
                                break;
                            }
                        case "Even":
                            {
                                comPort = new SerialPort(COMListSend.SelectedItem.ToString(),
Convert.ToInt32(SendBaud.Text), Parity.Even, 5, StopBits.One);
                                break;
                            }
                        case "Mark":
                            {
                                comPort = new SerialPort(COMListSend.SelectedItem.ToString(),
Convert.ToInt32(SendBaud.Text), Parity.Mark, 5, StopBits.One);
                                break;
                            }
                        case "Space":
                            {
                                comPort = new SerialPort(COMListSend.SelectedItem.ToString(),
Convert.ToInt32(SendBaud.Text), Parity.Space, 5, StopBits.One);
                                break;
                            }
                    }
                }
                comPort.ReadTimeout = 1000;
                comPort.WriteTimeout = 1000;
                comPort.DataReceived += SerialIventSecond;
                SendPotocol = SendProtocol.Text;
                if (SendPotocol == "ASCII")
                    comPort.Encoding = Encoding.ASCII;
                if (SendPotocol == "UTF-8")
                    comPort.Encoding = Encoding.UTF8;

                if (comPort.IsOpen)
                    MessageBox.Show("Порт с таким именем уже открыт");
                else
                {
                    comPort.Open();
                    Send.IsEnabled = true;
                    SendTwo.IsEnabled = true;
                    CloseSendPort.IsEnabled = true;
```

```
            }
        }
    }


    public void RecivePort(object sender, RoutedEventArgs e)
    {
        if (ReciveBaud.Text == "")
            MessageBox.Show("Установите скорость передачи данных в бодах");
        else
        {
            if (!(bool)ReciveParityCheck.IsChecked)
                comPort1 = new SerialPort(COMListRecive.SelectedItem.ToString(),
Convert.ToInt32(ReciveBaud.Text), Parity.None, 5, StopBits.One);
            else
            {
                switch (ParityRecive.Text.Split(" ")[0])
                {
                    case "Odd":
                        {
                            comPort1 = new SerialPort(COMListSend.SelectedItem.ToString(),
Convert.ToInt32(ReciveBaud.Text), Parity.Odd, 5, StopBits.One);
                            break;
                        }
                    case "Even":
                        {
                            comPort1 = new SerialPort(COMListSend.SelectedItem.ToString(),
Convert.ToInt32(ReciveBaud.Text), Parity.Even, 5, StopBits.One);
                            break;
                        }
                    case "Mark":
                        {
                            comPort1 = new SerialPort(COMListSend.SelectedItem.ToString(),
Convert.ToInt32(ReciveBaud.Text), Parity.Mark, 5, StopBits.One);
                            break;
                        }
                    case "Space":
                        {
                            comPort1 = new SerialPort(COMListSend.SelectedItem.ToString(),
Convert.ToInt32(ReciveBaud.Text), Parity.Space, 5, StopBits.One);
                            break;
                        }
                }
            }
            comPort1.ReadTimeout = 1000;
            comPort1.WriteTimeout = 1000;
```

```
        comPort1.DataReceived += SerialIventFirts;
        RecivePotocol = ReciveProtocol.Text;
        if (RecivePotocol == "ASCII")
            comPort1.Encoding = Encoding.ASCII;
        if (RecivePotocol == "UTF-8")
            comPort1.Encoding = Encoding.UTF8;

        if (comPort1.IsOpen)
            MessageBox.Show("Порт с таким именем уже открыт");
        else
        {
            comPort1.Open();
            Recive.IsEnabled = true;
            ReciveTwo.IsEnabled = true;
            CloseRecivePort.IsEnabled = true;
        }
    }
}

public void SendMessOne(object sender, RoutedEventArgs e)
{
    var mess = InPut.Text;
    if (SendProtocol.Text == "Binary" && COMPortUtil.Numform(mess))
    {
        switch (SendSummProt.Text)
        {
            case "None":
                {
                    var b = Convert.ToInt16(mess);
                    comPort.Write(BitConverter.GetBytes(b), 0, BitConverter.GetBytes(b).Length);
                    break;
                }
            case "Simple":
                {
                    var buff = BitConverter.GetBytes(Convert.ToInt16(mess));
                    sendSimple = new byte[1] { COMPortUtil.SimpleSumm(buff) };
                    SendContSumm.Text = comPort.Encoding.GetString(sendSimple);
                    var final = buff.Concat(sendSimple).ToArray();
                    comPort.Write(final, 0, final.Length);
                    break;
                }
            case "LRC":
                {
```

```
                    var buff = BitConverter.GetBytes(Convert.ToInt16(mess));
                    sendLRC = new byte[1] { COMPortUtil.LRC(buff) };
                    SendContSumm.Text = comPort.Encoding.GetString(sendLRC);
                    var final = buff.Concat(sendLRC).ToArray();
                    comPort.Write(final, 0, final.Length);
                    break;
                }
            case "CRC16":
                {
                    var buff = BitConverter.GetBytes(Convert.ToInt16(mess));
                    sendCRC16 = COMPortUtil.CRC16(buff);
                    SendContSumm.Text = comPort.Encoding.GetString(sendCRC16);
                    var final = buff.Concat(sendCRC16).ToArray();
                    comPort.Write(final, 0, final.Length);
                    break;
                }
            case "CRC32":
                {
                    var buff = BitConverter.GetBytes(Convert.ToInt16(mess));
                    sendCRC32 = COMPortUtil.CRC32(buff);
                    SendContSumm.Text = comPort.Encoding.GetString(sendCRC32);
                    var final = buff.Concat(sendCRC32).ToArray();
                    comPort.Write(final, 0, final.Length);
                    break;
                }
        }
    }
    else
    {
        switch (SendSummProt.Text)
        {
            case "None":
                {
                    comPort.Write(comPort.Encoding.GetBytes(mess), 0,
comPort.Encoding.GetBytes(mess).Length);
                    break;
                }
            case "Simple":
                {
                    var buff = comPort.Encoding.GetBytes(mess);
                    sendSimple = new byte[1] { COMPortUtil.SimpleSumm(buff) };
                    SendContSumm.Text = comPort1.Encoding.GetString(sendSimple);
                    var final = buff.Concat(sendSimple).ToArray();
```

```
                    comPort.Write(final, 0, final.Length);
                    break;
                }
            case "LRC":
                {
                    var buff = comPort.Encoding.GetBytes(mess);
                    sendLRC = new byte[1] { COMPortUtil.LRC(buff) };
                    SendContSumm.Text = comPort.Encoding.GetString(sendLRC);
                    var final = buff.Concat(sendLRC).ToArray();
                    comPort.Write(final, 0, final.Length);
                    break;
                }
            case "CRC16":
                {
                    var buff = comPort.Encoding.GetBytes(mess);
                    sendCRC16 = COMPortUtil.CRC16(buff);
                    SendContSumm.Text = comPort.Encoding.GetString(sendCRC16);
                    var final = buff.Concat(sendCRC16).ToArray();
                    comPort.Write(final, 0, final.Length);
                    break;
                }
            case "CRC32":
                {
                    var buff = comPort.Encoding.GetBytes(mess);
                    sendCRC32 = COMPortUtil.CRC32(buff);
                    SendContSumm.Text = comPort.Encoding.GetString(sendCRC32);
                    var final = buff.Concat(sendCRC32).ToArray();
                    comPort.Write(final, 0, final.Length);
                    break;
                }
        }
    }
}

public void SendMessTwo(object sender, RoutedEventArgs e)
{
    var mess = InPutSecond.Text;
    if (SendProtocol.Text == "Binary" && COMPortUtil.Numform(mess))
    {
        switch (SendSummProt.Text)
        {
            case "None":
                {
```

```
                var b = Convert.ToInt16(mess);
                comPort1.Write(BitConverter.GetBytes(b), 0, BitConverter.GetBytes(b).Length);
                break;
            }
        case "Simple":
            {
                var buff = BitConverter.GetBytes(Convert.ToInt16(mess));
                sendSimple = new byte[1] { COMPortUtil.SimpleSumm(buff) };
                SendContSummTwo.Text = comPort1.Encoding.GetString(sendSimple);
                var final = buff.Concat(sendSimple).ToArray();
                comPort1.Write(final, 0, final.Length);
                break;
            }
        case "LRC":
            {
                var buff = BitConverter.GetBytes(Convert.ToInt16(mess));
                sendLRC = new byte[1] { COMPortUtil.LRC(buff) };
                SendContSummTwo.Text = comPort1.Encoding.GetString(sendLRC);
                var final = buff.Concat(sendLRC).ToArray();
                comPort1.Write(final, 0, final.Length);
                break;
            }
        case "CRC16":
            {
                var buff = BitConverter.GetBytes(Convert.ToInt16(mess));
                sendCRC16 = COMPortUtil.CRC16(buff);
                SendContSummTwo.Text = comPort1.Encoding.GetString(sendCRC16);
                var final = buff.Concat(sendCRC16).ToArray();
                comPort1.Write(final, 0, final.Length);
                break;
            }
        case "CRC32":
            {
                var buff = BitConverter.GetBytes(Convert.ToInt16(mess));
                sendCRC32 = COMPortUtil.CRC32(buff);
                SendContSummTwo.Text = comPort1.Encoding.GetString(sendCRC32);
                var final = buff.Concat(sendCRC32).ToArray();
                comPort1.Write(final, 0, final.Length);
                break;
            }
    }
}
else
```

```
            {
                switch (SendSummProt.Text)
                {
                    case "None":
                        {
                            comPort1.Write(comPort1.Encoding.GetBytes(mess), 0,
comPort1.Encoding.GetBytes(mess).Length);
                            break;
                        }
                    case "Simple":
                        {
                            var buff = comPort.Encoding.GetBytes(mess);
                            sendSimple = new byte[1] { COMPortUtil.SimpleSumm(buff) };
                            SendContSummTwo.Text = comPort1.Encoding.GetString(sendSimple);
                            var final = buff.Concat(sendSimple).ToArray();
                            comPort1.Write(final, 0, final.Length);
                            break;
                        }
                    case "LRC":
                        {
                            var buff = comPort1.Encoding.GetBytes(mess);
                            sendLRC = new byte[1] { COMPortUtil.LRC(buff) };
                            SendContSummTwo.Text = comPort1.Encoding.GetString(sendLRC);
                            var final = buff.Concat(sendLRC).ToArray();
                            comPort1.Write(final, 0, final.Length);
                            break;
                        }
                    case "CRC16":
                        {
                            var buff = comPort1.Encoding.GetBytes(mess);
                            sendCRC16 = COMPortUtil.CRC16(buff);
                            SendContSummTwo.Text = comPort1.Encoding.GetString(sendCRC16);
                            var final = buff.Concat(sendCRC16).ToArray();
                            comPort1.Write(final, 0, final.Length);
                            break;
                        }
                    case "CRC32":
                        {
                            var buff = comPort1.Encoding.GetBytes(mess);
                            sendCRC32 = COMPortUtil.CRC32(buff);
                            SendContSummTwo.Text = comPort1.Encoding.GetString(sendCRC32);
                            var final = buff.Concat(sendCRC32).ToArray();
                            comPort1.Write(final, 0, final.Length);
```

```
                    break;
                }
        }
    }
}

public void SerialIventFirts(object sender, SerialDataReceivedEventArgs e)
{
    recivebyte = new byte[comPort1.BytesToRead];
    comPort1.Read(recivebyte, 0, comPort1.BytesToRead);
}

public void SerialIventSecond(object sender, SerialDataReceivedEventArgs e)
{
    recivebyte = new byte[comPort.BytesToRead];
    comPort.Read(recivebyte, 0, comPort.BytesToRead);
}

public void ReciveMessOne(object sender, RoutedEventArgs e)
{
    if (ReciveProtocol.Text == "Binary")
    {
        switch (ReciveSummProt.Text)
        {
            case "None":
                {
                    var buff = BitConverter.ToInt16(recivebyte);
                    OutPutMess.Text = buff.ToString();
                    break;
                }
            case "Simple":
                {
                    byte[] buff = new byte[recivebyte.Length - 1];
                    Array.Copy(recivebyte, buff, recivebyte.Length - 1);


                    if (ModBitOne == true)
                        buff = COMPortUtil.ModByte(buff);

                    if (SwapBitOne == true)
                        if (buff.Length > 1)
                            buff = COMPortUtil.SwapByte(buff, 0, 1);
                        else
```

```
                MessageBox.Show("Длина сообщения слишком мала");


        reciveSimple = new byte[1] { COMPortUtil.SimpleSumm(buff) };
        var b = BitConverter.ToInt16(buff);
        ReciveContSumm.Text = comPort.Encoding.GetString(reciveLRC);
        if (reciveSimple.SequenceEqual(sendSimple))
            OutPutMess.Text = b.ToString();
        else
            MessageBox.Show("Контрольная сумма не совпала");
        break;
    }
case "LRC":
    {
        byte[] buff = new byte[recivebyte.Length - 1];
        Array.Copy(recivebyte, buff, recivebyte.Length - 1);



        if (ModBitOne == true)
            buff = COMPortUtil.ModByte(buff);


        if (SwapBitOne == true)
            if (buff.Length > 1)
                buff = COMPortUtil.SwapByte(buff, 0, 1);
            else
                MessageBox.Show("Длина сообщения слишком мала");


        reciveLRC = new byte[1] { COMPortUtil.LRC(buff) };
        var b = BitConverter.ToInt16(buff);
        ReciveContSumm.Text = comPort.Encoding.GetString(reciveLRC);
        if (reciveLRC.SequenceEqual(sendLRC))
            OutPutMess.Text = b.ToString();
        else
            MessageBox.Show("Контрольная сумма не совпала");
        break;
    }
case "CRC16":
    {
        byte[] buff = new byte[recivebyte.Length - 2];
        Array.Copy(recivebyte, buff, recivebyte.Length - 2);


        if (ModBitOne == true)
            buff = COMPortUtil.ModByte(buff);
```

```
                if (SwapBitOne == true)
                    if (buff.Length > 1)
                        buff = COMPortUtil.SwapByte(buff, 0, 1);
                    else
                        MessageBox.Show("Длина сообщения слишком мала");

                reciveCRC16 = COMPortUtil.CRC16(buff);
                var b = BitConverter.ToInt16(buff);
                ReciveContSumm.Text = comPort.Encoding.GetString(reciveCRC16);
                if (reciveCRC16.SequenceEqual(sendCRC16))
                    OutPutMess.Text = b.ToString();
                else
                    MessageBox.Show("Контрольная сумма не совпала");
                break;
            }
        case "CRC32":
            {
                byte[] buff = new byte[recivebyte.Length - 4];
                Array.Copy(recivebyte, buff, recivebyte.Length - 4);

                if (ModBitOne == true)
                    buff = COMPortUtil.ModByte(buff);

                if (SwapBitOne == true)
                    if (buff.Length > 1)
                        buff = COMPortUtil.SwapByte(buff, 0, 1);
                    else
                        MessageBox.Show("Длина сообщения слишком мала");

                reciveCRC32 = COMPortUtil.CRC32(buff);
                var b = BitConverter.ToInt16(buff);
                ReciveContSumm.Text = comPort.Encoding.GetString(reciveCRC32);
                if (reciveCRC32.SequenceEqual(sendCRC32))
                    OutPutMess.Text = b.ToString();
                else
                    MessageBox.Show("Контрольная сумма не совпала");
                break;
            }
    }
}
else
{
    switch (ReciveSummProt.Text)
```

```
{
    case "None":
        {
            OutPutMess.Text = comPort.Encoding.GetString(recivebyte);
            break;
        }
    case "Simple":
        {
            byte[] buff = new byte[recivebyte.Length - 1];
            Array.Copy(recivebyte, buff, recivebyte.Length - 1);

            if (ModBitOne == true)
                buff = COMPortUtil.ModByte(buff);

            if (SwapBitOne == true)
                if (buff.Length > 1)
                    buff = COMPortUtil.SwapByte(buff, 0, 1);
                else
                    MessageBox.Show("Длина сообщения слишком мала");

            reciveLRC = new byte[1] { COMPortUtil.SimpleSumm(buff) };
            ReciveContSumm.Text = comPort.Encoding.GetString(reciveLRC);
            if (reciveLRC.SequenceEqual(sendSimple))
                OutPutMess.Text = comPort.Encoding.GetString(buff);
            else
                MessageBox.Show("Контрольная сумма не совпала");
            break;
        }
    case "LRC":
        {
            byte[] buff = new byte[recivebyte.Length - 1];
            Array.Copy(recivebyte, buff, recivebyte.Length - 1);

            if (ModBitOne == true)
                buff = COMPortUtil.ModByte(buff);

            if (SwapBitOne == true)
                if (buff.Length > 1)
                    buff = COMPortUtil.SwapByte(buff, 0, 1);
                else
                    MessageBox.Show("Длина сообщения слишком мала");

            reciveLRC = new byte[1] { COMPortUtil.LRC(buff) };
```

```
            ReciveContSumm.Text = comPort.Encoding.GetString(reciveLRC);
            if (reciveLRC.SequenceEqual(sendLRC))
                OutPutMess.Text = comPort.Encoding.GetString(buff);
            else
                MessageBox.Show("Контрольная сумма не совпала");
            break;
        }
    case "CRC16":
        {
            byte[] buff = new byte[recivebyte.Length - 2];
            Array.Copy(recivebyte, buff, recivebyte.Length - 2);

            if (ModBitOne == true)
                buff = COMPortUtil.ModByte(buff);

            if (SwapBitOne == true)
                if (buff.Length > 1)
                    buff = COMPortUtil.SwapByte(buff, 0, 1);
                else
                    MessageBox.Show("Длина сообщения слишком мала");

            reciveCRC16 = COMPortUtil.CRC16(buff);
            ReciveContSumm.Text = comPort.Encoding.GetString(reciveCRC16);
            if (reciveCRC16.SequenceEqual(sendCRC16))
                OutPutMess.Text = comPort.Encoding.GetString(buff);
            else
                MessageBox.Show("Контрольная сумма не совпала");
            break;
        }
    case "CRC32":
        {
            byte[] buff = new byte[recivebyte.Length - 4];
            Array.Copy(recivebyte, buff, recivebyte.Length - 4);

            if (ModBitOne == true)
                buff = COMPortUtil.ModByte(buff);

            if (SwapBitOne == true)
                if (buff.Length > 1)
                    buff = COMPortUtil.SwapByte(buff, 0, 1);
                else
                    MessageBox.Show("Длина сообщения слишком мала");
```

```
                    reciveCRC32 = COMPortUtil.CRC32(buff);
                    ReciveContSumm.Text = comPort.Encoding.GetString(reciveCRC32);
                    if (reciveCRC32.SequenceEqual(sendCRC32))
                        OutPutMess.Text = comPort.Encoding.GetString(buff);
                    else
                        MessageBox.Show("Контрольная сумма не совпала");
                    break;
                }
            }
        }
}



public void ReciveMessTwo(object sender, RoutedEventArgs e)
{
    if(ReciveProtocol.Text == "Binary")
    {
        switch (ReciveSummProt.Text)
        {
            case "None":
                {
                    var buff = BitConverter.ToInt16(recivebyte);
                    OutPutMessSecond.Text = buff.ToString();
                    break;
                }
            case "Simple":
                {
                    byte[] buff = new byte[recivebyte.Length - 1];
                    Array.Copy(recivebyte, buff, recivebyte.Length - 1);

                    if (ModBitTwo == true)
                        buff = COMPortUtil.ModByte(buff);

                    if (SwapBitTwo == true)
                        if (buff.Length > 1)
                            buff = COMPortUtil.SwapByte(buff, 0, 1);
                        else
                            MessageBox.Show("Длина сообщения слишком мала");

                    reciveSimple = new byte[1] { COMPortUtil.SimpleSumm(buff) };
                    var b = BitConverter.ToInt16(buff);
                    ReciveContSummTwo.Text = comPort.Encoding.GetString(reciveSimple);
                    if (reciveSimple.SequenceEqual(sendSimple))
```

```
            OutPutMessSecond.Text = b.ToString();
        else
            MessageBox.Show("Контрольная сумма не совпала");
        break;
    }
case "LRC":
    {
        byte[] buff = new byte[recivebyte.Length - 1];
        Array.Copy(recivebyte, buff, recivebyte.Length - 1);

        if (ModBitTwo == true)
            buff = COMPortUtil.ModByte(buff);

        if (SwapBitTwo == true)
            if (buff.Length > 1)
                buff = COMPortUtil.SwapByte(buff, 0, 1);
            else
                MessageBox.Show("Длина сообщения слишком мала");

        reciveLRC = new byte[1] { COMPortUtil.LRC(buff) };
        var b = BitConverter.ToInt16(buff);
        ReciveContSummTwo.Text = comPort.Encoding.GetString(reciveLRC);
        if (reciveLRC.SequenceEqual(sendLRC))
            OutPutMessSecond.Text = b.ToString();
        else
            MessageBox.Show("Контрольная сумма не совпала");
        break;
    }
case "CRC16":
    {
        byte[] buff = new byte[recivebyte.Length - 2];
        Array.Copy(recivebyte, buff, recivebyte.Length - 2);

        if (ModBitTwo == true)
            buff = COMPortUtil.ModByte(buff);

        if (SwapBitTwo == true)
            if (buff.Length > 1)
                buff = COMPortUtil.SwapByte(buff, 0, 1);
            else
                MessageBox.Show("Длина сообщения слишком мала");

        reciveCRC16 = COMPortUtil.CRC16(buff);
```

```
                var b = BitConverter.ToInt16(buff);
                ReciveContSummTwo.Text = comPort.Encoding.GetString(reciveCRC16);
                if (reciveCRC16.SequenceEqual(sendCRC16))
                    OutPutMessSecond.Text = b.ToString();
                else
                    MessageBox.Show("Контрольная сумма не совпала");
                break;
            }
        case "CRC32":
            {
                byte[] buff = new byte[recivebyte.Length - 4];
                Array.Copy(recivebyte, buff, recivebyte.Length - 4);

                if (ModBitTwo == true)
                    buff = COMPortUtil.ModByte(buff);

                if (SwapBitTwo == true)
                    if (buff.Length > 1)
                        buff = COMPortUtil.SwapByte(buff, 0, 1);
                    else
                        MessageBox.Show("Длина сообщения слишком мала");

                reciveCRC32 = COMPortUtil.CRC32(buff);
                var b = BitConverter.ToInt16(buff);
                ReciveContSummTwo.Text = comPort.Encoding.GetString(reciveCRC32);
                if (reciveCRC32.SequenceEqual(sendCRC32))
                    OutPutMessSecond.Text = b.ToString();
                else
                    MessageBox.Show("Контрольная сумма не совпала");
                break;
            }
    }
}
else
{
    switch (ReciveSummProt.Text)
    {
        case "None":
            {
                OutPutMessSecond.Text = comPort.Encoding.GetString(recivebyte);
                break;
            }
        case "Simple":
```

```
    {
        byte[] buff = new byte[recivebyte.Length - 1];
        Array.Copy(recivebyte, buff, recivebyte.Length - 1);

        if (ModBitTwo == true)
            buff = COMPortUtil.ModByte(buff);

        if (SwapBitTwo == true)
            if (buff.Length > 1)
                buff = COMPortUtil.SwapByte(buff, 0, 1);
            else
                MessageBox.Show("Длина сообщения слишком мала");

        reciveSimple = new byte[1] { COMPortUtil.SimpleSumm(buff) };
        ReciveContSummTwo.Text = comPort.Encoding.GetString(reciveSimple);
        if (reciveSimple.SequenceEqual(sendSimple))
            OutPutMessSecond.Text = comPort.Encoding.GetString(buff);
        else
            MessageBox.Show("Контрольная сумма не совпала");
        break;
    }
case "LRC":
    {
        byte[] buff = new byte[recivebyte.Length - 1];
        Array.Copy(recivebyte, buff, recivebyte.Length - 1);

        if (ModBitTwo == true)
            buff = COMPortUtil.ModByte(buff);

        if (SwapBitTwo == true)
            if (buff.Length > 1)
                buff = COMPortUtil.SwapByte(buff, 0, 1);
            else
                MessageBox.Show("Длина сообщения слишком мала");

        reciveLRC = new byte[1] { COMPortUtil.LRC(buff) };
        ReciveContSummTwo.Text = comPort.Encoding.GetString(reciveLRC);
        if (reciveLRC.SequenceEqual(sendLRC))
            OutPutMessSecond.Text = comPort.Encoding.GetString(buff);
        else
            MessageBox.Show("Контрольная сумма не совпала");
        break;
    }
```

```
case "CRC16":
    {
        byte[] buff = new byte[recivebyte.Length - 2];
        Array.Copy(recivebyte, buff, recivebyte.Length - 2);

        if (ModBitTwo == true)
            buff = COMPortUtil.ModByte(buff);

        if (SwapBitTwo == true)
            if (buff.Length > 1)
                buff = COMPortUtil.SwapByte(buff, 0, 1);
            else
                MessageBox.Show("Длина сообщения слишком мала");

        reciveCRC16 = COMPortUtil.CRC16(buff);
        ReciveContSummTwo.Text = comPort.Encoding.GetString(reciveCRC16);
        if (reciveCRC16.SequenceEqual(sendCRC16))
            OutPutMessSecond.Text = comPort.Encoding.GetString(buff);
        else
            MessageBox.Show("Контрольная сумма не совпала");
        break;
    }
case "CRC32":
    {
        byte[] buff = new byte[recivebyte.Length - 4];
        Array.Copy(recivebyte, buff, recivebyte.Length - 4);

        if (ModBitTwo == true)
            buff = COMPortUtil.ModByte(buff);

        if (SwapBitTwo == true)
            if (buff.Length > 1)
                buff = COMPortUtil.SwapByte(buff, 0, 1);
            else
                MessageBox.Show("Длина сообщения слишком мала");

        reciveCRC32 = COMPortUtil.CRC32(buff);
        ReciveContSummTwo.Text = comPort.Encoding.GetString(reciveCRC32);
        if (reciveCRC32.SequenceEqual(sendCRC32))
            OutPutMessSecond.Text = comPort.Encoding.GetString(buff);
        else
            MessageBox.Show("Контрольная сумма не совпала");
        break;
```

```
                }
            }
        }
    }
private void CheckBox_Checked(object sender, RoutedEventArgs e)
{
    ParitySend.Visibility = Visibility.Visible;
    ParitySendLabel.Visibility = Visibility.Visible;
}

private void CheckBox_Checked_1(object sender, RoutedEventArgs e)
{
    ParityRecive.Visibility = Visibility.Visible;
    ParityReciveLabel.Visibility = Visibility.Visible;
}

private void SendPortClose(object sender, RoutedEventArgs e)
{
    comPort.Close();
    CloseSendPort.IsEnabled = false;
    SendBaud.Clear();
}

private void RecivePortClose(object sender, RoutedEventArgs e)
{
    comPort1.Close();
    CloseRecivePort.IsEnabled = false;
    ReciveBaud.Clear();
}

private void ModByteReciveTwo_Checked(object sender, RoutedEventArgs e)
{
    ModBitOne = true;
}

private void ModByteReciveOne_Checked(object sender, RoutedEventArgs e)
{
    ModBitTwo = true;
}

private void SwapCheckOne_Checked(object sender, RoutedEventArgs e)
{
    SwapBitOne = true;
```

```
        }

        private void SwapCheckTwo_Checked(object sender, RoutedEventArgs e)
        {
            SwapBitTwo = true;
        }
    }
```

COMPortUtil.cs

```
public static class COMPortUtil
    {
        private const string AllowedNum= "0123456789";

        public static byte LRC(byte[] bytes)
        {
            int LRC = 0;
            for (int i = 0; i < bytes.Length; i++)
            {
                LRC -= bytes[i];
            }
            return (byte)LRC;
        }

        public static void Request(SerialPort comPort, string text, string format = "text")
        {
            switch(format)
            {
                case "text":
                    {
                        WriteString(text, comPort);
                        break;
                    }
                case "byte":
                    {
                        WriteByte(text, comPort);
                        break;
                    }
            }
        }

        public static void Dispatch(SerialPort comPort, string text, string format = "text")
```

```
{
    switch (format)
    {
        case "text":
            {
                WriteString(text, comPort);
                break;
            }
        case "byte":
            {
                WriteByte(text, comPort);
                break;
            }
    }
}

public static string Receive(SerialPort comPort)
{
    return ReadText(comPort);
}

public static byte ToByte(this BitArray bits)
{
    if (bits.Count != 8)
    {
        throw new ArgumentException("bits");
    }
    byte[] bytes = new byte[1];
    bits.CopyTo(bytes, 0);
    return bytes[0];
}

public static byte SimpleSumm(byte[] data)
{
    byte[] start = new byte[1] { data[0] };
    var buff = new BitArray(start);
    foreach(var i in data.Skip(1))
    {
        byte[] ibuff = new byte[1] { i };
        buff = buff.Xor(new BitArray(ibuff));
    }
    var output = buff.ToByte();
    return output;
```

```
}

public static ushort ComputeChecksumCRC16(byte[] bytes)
{
    ushort crc = 0xFFFF;
    const ushort poly = 0x1021;
    ushort[] table = new ushort[256];
    ushort temp, a;
    for (int i = 0; i < table.Length; ++i)
    {
        temp = 0;
        a = (ushort)(i << 8);
        for (int j = 0; j < 8; ++j)
        {
            if (((temp ^ a) & 0x8000) != 0)
            {
                temp = (ushort)((temp << 1) ^ poly);
            }
            else
            {
                temp <<= 1;
            }
            a <<= 1;
        }
        table[i] = temp;
    }

    foreach (var it in bytes)
    {
        crc = (ushort)((crc << 8) ^ table[((crc >> 8) ^ (0xff & it))]);
    }

    return crc;
}

public static byte[] CRC16(byte[] bytes)
{
    ushort crc = ComputeChecksumCRC16(bytes);
    return BitConverter.GetBytes(crc);
}



static uint Crc32(byte[] array)
```

```
{
    uint[] crc_table = new uint[256];

    uint result = 0xFFFFFFFF;
    uint crc;

    for (uint i = 0; i < 256; i++)
    {
        crc = i;
        for (int j = 0; j < 8; j++)
            crc = ((crc & 1) == 1) ? (crc >> 1) ^ 0xEDB88320 : crc >> 1;

        crc_table[i] = crc;
    }

    for (int i = 0; i < array.Length; i++)
    {
        byte last_byte = (byte)(result & 0xFF);
        result >>= 8;
        result = result ^ crc_table[last_byte ^ array[i]];
    }

    return result;
}

public static byte[] CRC32(byte[] bytes)
{
    uint crc = Crc32(bytes);
    return BitConverter.GetBytes(crc);
}

public static bool Numform(string buff)
{
    bool flag = false;
    foreach (var i in buff)
    {
        flag = false;
        foreach (var j in AllowedNum)
        {
            if (i == j)
            {
                flag = true;
                break;
```

```
                }
            }
            if (!flag)
                break;
        }
        return flag;
    }

    public static byte[] ModByte(byte[] data)
    {
        Random rnd = new Random((int)DateTime.Now.Ticks);
        var bt = new Byte[1];
        rnd.NextBytes(bt);
        var output = data;
        output[rnd.Next(data.Length)] = bt[0];
        return output;
    }

    public static byte[] SwapByte(byte[] data, int num1, int num2)
    {
        var output = data;
        var b = output[num1];
        output[num1] = output[num2];
        output[num2] = b;
        return output;
    }
}
```