Московский государственный технический университет им. Н.Э. Баумана
Кафедра «Системы обработки информации и управления»

УТВЕРЖДАЮ:

_____ Галкин В. А.

«__» _____ 2017 г.

Листинг программы
к курсовой работе
«Локальная безадаптерная сеть»
(вариант № 26а)
по курсу «Сетевые технологии в АСОИУ»

ИСПОЛНИТЕЛИ:

_____ Лещев А. О., ИУ5-64

_____ Мельников К. И., ИУ5-64

«__» _____ 2017 г.

Москва — 2017 г.

Содержание

1. Файл App.xaml

```
<Application x:Class="iu5nt.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    StartupUri="MainWindow.xaml">
  <Application.Resources>

  </Application.Resources>
</Application>
```

2. Файл App.xaml.cs

```
using System.Windows;

namespace iu5nt
{
  public partial class App : Application
  {
  }
}
```

3. Файл MainWindow.xaml

```
<Window x:Class="iu5nt.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Локальная безадаптерная сеть" Height="300" Width="400"
    MinHeight="300" MinWidth="400">
  <Grid>
    <Grid.RowDefinitions>
      <RowDefinition Height="Auto"/>
      <RowDefinition Height="Auto"/>
      <RowDefinition Height="Auto"/>
      <RowDefinition Height="Auto"/>
      <RowDefinition/>
    </Grid.RowDefinitions>
    <GroupBox x:Name="ConnectionBox" Header="Физическое соединение"
      Margin="9 2 9 3">
      <Grid>
        <Grid.RowDefinitions>
          <RowDefinition Height="Auto"/>
          <RowDefinition/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
          <ColumnDefinition Width="Auto"/>
          <ColumnDefinition/>
          <ColumnDefinition Width="Auto"/>
          <ColumnDefinition Width="Auto"/>
```

```
26          </Grid.ColumnDefinitions>
27          <ComboBox x:Name="PortsList" Margin="3 4 0 6" Grid.ColumnSpan="2"/>
28          <Button x:Name="OpenButton" Content="Открыть порт"
29             Margin="7 4 0 6" MinWidth="94" MinHeight="23" Grid.Column="2"
30            Click="OpenButton_Click"/>
31          <Button x:Name="CloseButton" Content="Закрыть порт" IsEnabled="False"
32            Margin="7 4 2 6" MinWidth="94" MinHeight="23" Grid.Column="3"
33            Click="CloseButton_Click"/>
34          <RadioButton x:Name="DtrIndicator" Content="DTR" GroupName="DTR"
35            Margin="3 4 0 6" MinWidth="60" Grid.Row="1" IsEnabled="False"/>
36          <RadioButton x:Name="DsrIndicator" Content="DSR" GroupName="DSR"
37            Margin="7 4 0 6" Grid.Row="1" Grid.Column="1" IsEnabled="False"/>
38          <RadioButton x:Name="RtsIndicator" Content="RTS" GroupName="RTS"
39            Margin="7 4 0 6" Grid.Row="1" Grid.Column="2" IsEnabled="False"/>
40          <RadioButton x:Name="CtsIndicator" Content="CTS" GroupName="CTS"
41            Margin="7 4 2 6" Grid.Row="1" Grid.Column="3" IsEnabled="False"/>
42        </Grid>
43      </GroupBox>
44      <GroupBox Header="Отправка файла" x:Name="FileBox" IsEnabled="False"
45        Margin="9 0 9 3" Grid.Row="1">
46        <Grid>
47          <Grid.ColumnDefinitions>
48            <ColumnDefinition/>
49            <ColumnDefinition Width="Auto"/>
50            <ColumnDefinition Width="Auto"/>
51          </Grid.ColumnDefinitions>
52          <TextBox x:Name="FileName" IsReadOnly="True" Margin="3 4 0 6"/>
53          <Button x:Name="SelectFile" Content="Выбрать файл"
54            Margin="7 4 0 6" MinWidth="94" MinHeight="23" Grid.Column="1"
55            Click="SelectFile_Click"/>
56          <Button x:Name="SendFile" Content="Отправить файл" IsEnabled="False"
57            Margin="7 4 2 6" MinWidth="94" MinHeight="23" Grid.Column="2"
58            Click="SendFile_Click"/>
59        </Grid>
60      </GroupBox>
61      <GroupBox Header="Приём файла" x:Name="DirectoryBox" IsEnabled="False"
62        Margin="9 0 9 3" Grid.Row="2">
63        <Grid>
64          <Grid.ColumnDefinitions>
65            <ColumnDefinition/>
66            <ColumnDefinition Width="Auto"/>
67          </Grid.ColumnDefinitions>
68          <TextBox x:Name="DirectoryName" IsReadOnly="True" Margin="3 4 0 6"/>
69          <Button x:Name="SelectDirectory" Content="Выбрать папку"
70            Margin="7 4 2 6" MinWidth="94" MinHeight="23" Grid.Column="1"
71            Click="SelectDirectory_Click"/>
72        </Grid>
73      </GroupBox>
74      <TextBlock x:Name="StatusText" Text="Соединение не установлено."
75        TextWrapping="WrapWithOverflow" TextAlignment="Center" Margin="0"
```

```
76          Grid.Row="3"/>
77        <Grid Grid.Row="4">
78          <Grid.ColumnDefinitions>
79            <ColumnDefinition/>
80            <ColumnDefinition Width="Auto"/>
81          </Grid.ColumnDefinitions>
82          <ProgressBar x:Name="ProgressBar" Maximum="1" Margin="11 7 0 11"/>
83          <Button x:Name="DisconnectButton" Content="Разъединить"
84            IsEnabled="False" Margin="7 7 11 11" Grid.Column="1"
85            Click="DisconnectButton_Click"/>
86        </Grid>
87      </Grid>
88    </Window>
```

4. Файл MainWindow.xaml.cs

```
1   using System;
2   using System.IO;
3   using System.IO.Ports;
4   using System.Security.Cryptography;
5   using System.Windows;
6   using System.Windows.Threading;
7   using wf = System.Windows.Forms;
8
9   namespace iu5nt
10  {
11    public partial class MainWindow : Window, IDisposable
12    {
13      private wf.OpenFileDialog fileDialog = new wf.OpenFileDialog();
14      private wf.FolderBrowserDialog folderDialog = new wf.FolderBrowserDialog();
15
16      private bool folderReady = false;
17      private bool? sending = null;
18      private Stream fileStream;
19      private string fileName, hashName, filePath, tempPath;
20      private long length;
21      private const short chunkSize = 512;
22
23      private DispatcherTimer timer = new DispatcherTimer() {
24        Interval = new TimeSpan(0, 0, 2) // 2 seconds
25      };
26      private ushort retries = 0;
27      private const ushort maxRetries = 3;
28      private byte[] lastPacket;
29
30      public MainWindow()
31      {
32        InitializeComponent();
33        PortsList.ItemsSource = SerialPort.GetPortNames();
34        timer.Tick += ResendPacket;
```

```
35          DataLink.OnRecieve += InvokeHandler;
36          Physical.OnCheck += PortCheck;
37          Physical.UICheck += PortCheck;
38          Dispatcher.UnhandledException += ExceptionHandler;
39        }
40
41        private void OpenButton_Click(object sender, RoutedEventArgs e)
42        {
43          var port = PortsList.SelectedItem;
44          if (port == null)
45          {
46            MessageBox.Show("Сначала необходимо выбрать порт.");
47          }
48          else
49          {
50            Physical.Connect((string)port);
51            if (folderReady)
52            {
53              Physical.SetRts(true);
54            }
55            OpenButton.IsEnabled = false;
56            CloseButton.IsEnabled = true;
57            PortsList.IsEnabled = false;
58            FileBox.IsEnabled = true;
59            DirectoryBox.IsEnabled = true;
60            StatusText.Text = "Физическое соединение открыто.";
61          }
62        }
63
64        private void CloseButton_Click(object sender, RoutedEventArgs e)
65        {
66          Physical.Disconnect();
67          DsrIndicator.IsChecked = false;
68          CtsIndicator.IsChecked = false;
69          OpenButton.IsEnabled = true;
70          CloseButton.IsEnabled = false;
71          PortsList.IsEnabled = true;
72          FileBox.IsEnabled = false;
73          DirectoryBox.IsEnabled = false;
74          DtrIndicator.IsChecked = false;
75          DsrIndicator.IsChecked = false;
76          RtsIndicator.IsChecked = false;
77          CtsIndicator.IsChecked = false;
78          StatusText.Text = "Физическое соединение закрыто.";
79        }
80
81        private void SelectFile_Click(object sender, RoutedEventArgs e)
82        {
83          var result = fileDialog.ShowDialog();
84          if (result == wf.DialogResult.OK)
```

```
85          {
86            FileName.Text = fileDialog.FileName;
87            SendFile.IsEnabled = true;
88            sending = null;
89          }
90        }
91
92        private void SelectDirectory_Click(object sender, RoutedEventArgs e)
93        {
94          var result = folderDialog.ShowDialog();
95          if (result == wf.DialogResult.OK)
96          {
97            DirectoryName.Text = folderDialog.SelectedPath;
98            folderReady = true;
99            Physical.SetRts(true);
100           sending = null;
101           StatusText.Text = "Ожидаем логического соединения...";
102         }
103       }
104
105       private void SendFile_Click(object sender, RoutedEventArgs e)
106       {
107         if (CtsIndicator.IsChecked != true || DsrIndicator.IsChecked != true)
108         {
109           MessageBox.Show(
110             "Принимающая сторона не готова к логическому соединению.");
111           return;
112         }
113
114         fileStream = fileDialog.OpenFile();
115         var hash = new SHA512CryptoServiceProvider().ComputeHash(fileStream);
116         fileStream.Seek(0, SeekOrigin.Begin);
117
118         var stream = new MemoryStream();
119         var writer = new BinaryWriter(stream);
120
121         writer.Write((byte)MessageType.FileName);
122         writer.Write(fileDialog.SafeFileName);
123         writer.Write(fileStream.Length);
124         writer.Write(hash); // 64 bytes for security
125
126         sending = true;
127         CloseButton.IsEnabled = false;
128         FileBox.IsEnabled = false;
129         DirectoryBox.IsEnabled = false;
130         StatusText.Text = "Установка логического соединения...";
131         Title = "Отправляем " + fileDialog.SafeFileName;
132         Physical.SetRts(true);
133         SendPacket(stream.ToArray());
134       }
```

```
135
136       private void DisconnectButton_Click(object sender, RoutedEventArgs e)
137       {
138         sending = null;
139         CloseButton.IsEnabled = true;
140         FileBox.IsEnabled = true;
141         DirectoryBox.IsEnabled = true;
142         DisconnectButton.IsEnabled = false;
143         if (fileStream != null)
144         {
145           fileStream.Close();
146         }
147
148         StatusText.Text = "Логическое соединение разорвано.";
149         SendPacket(new byte[] { (byte)MessageType.Disconnect });
150       }
151
152       private void InvokeHandler(byte[] packet, bool check)
153       {
154         Dispatcher.Invoke(new DataLink.RecieveMEthod(ReceiveMessage),
155           DispatcherPriority.Send, packet, check);
156       }
157
158       private void ReceiveMessage(byte[] packet, bool check)
159       {
160         if (!check)
161         {
162           MessageBox.Show("Получен повреждённый пакет.");
163           return;
164         }
165
166         var stream = new MemoryStream(packet);
167         var reader = new BinaryReader(stream);
168         switch ((MessageType)reader.ReadByte())
169         {
170           case MessageType.FileName:
171             ParseFileName(reader);
172             break;
173           case MessageType.ReceiveNotReady:
174             timer.Stop();
175             CloseButton.IsEnabled = true;
176             FileBox.IsEnabled = true;
177             DirectoryBox.IsEnabled = true;
178             StatusText.Text = "Физическое соединение открыто.";
179             MessageBox.Show(
180               "Принимающая сторона не готова к логическому соединению.");
181             break;
182           case MessageType.FileRequest:
183             SendFileChunk(reader);
184             break;
```

```
185           case MessageType.FileChunk:
186             SaveFileChunk(reader);
187             break;
188           case MessageType.FileReceived:
189             timer.Stop();
190             sending = null;
191             CloseButton.IsEnabled = true;
192             FileBox.IsEnabled = true;
193             DirectoryBox.IsEnabled = true;
194             DisconnectButton.IsEnabled = false;
195             StatusText.Text = "Файл успешно передан.";
196             Title = "Локальная безадаптерная сеть";
197             ProgressBar.Value = 1;
198             SendPacket(new byte[] { (byte)MessageType.FileReceivedOk });
199             Physical.SetRts(folderReady);
200             break;
201           case MessageType.FileReceivedOk:
202             if (sending == null) break;
203             timer.Stop();
204             sending = null;
205             CloseButton.IsEnabled = true;
206             FileBox.IsEnabled = true;
207             DirectoryBox.IsEnabled = true;
208             DisconnectButton.IsEnabled = false;
209             StatusText.Text = "Файл успешно получен.";
210             Title = "Локальная безадаптерная сеть";
211             break;
212           case MessageType.Disconnect:
213             timer.Stop();
214             if (fileStream != null)
215             {
216               fileStream.Close();
217             }
218             sending = null;
219             CloseButton.IsEnabled = true;
220             FileBox.IsEnabled = true;
221             DirectoryBox.IsEnabled = true;
222             DisconnectButton.IsEnabled = false;
223             StatusText.Text = "Логическое соединение разорвано.";
224             Title = "Локальная безадаптерная сеть";
225             SendPacket(new byte[] { (byte)MessageType.DisconnectOk });
226             Physical.SetRts(folderReady);
227             break;
228           case MessageType.DisconnectOk:
229             timer.Stop();
230             Physical.SetRts(folderReady);
231             break;
232           default:
233             MessageBox.Show("Получен неизвестный пакет.");
234             break;
```

```
235         }
236       }
237
238       private void ParseFileName(BinaryReader reader)
239       {
240         if (!folderReady)
241         {
242           SendPacket(new byte[] { (byte)MessageType.ReceiveNotReady });
243           timer.Stop();
244           MessageBox.Show("Необходимо выбрать папку для приёма файла.");
245           return;
246         }
247         fileName = reader.ReadString();
248         length = reader.ReadInt64();
249         var hash = reader.ReadBytes(64);
250
251         hashName = "";
252         foreach (var b in hash)
253         {
254           hashName += b.ToString("x2");
255         }
256
257         tempPath = Path.Combine(folderDialog.SelectedPath, hashName);
258         filePath = Path.Combine(folderDialog.SelectedPath, fileName);
259
260         fileStream = File.OpenWrite(tempPath);
261         fileStream.Seek(0, SeekOrigin.End);
262
263         DisconnectButton.IsEnabled = true;
264         CloseButton.IsEnabled = false;
265         FileBox.IsEnabled = false;
266         DirectoryBox.IsEnabled = false;
267         StatusText.Text = "Логическое соединение установлено.";
268         Title = "Принимаем " + fileName;
269         sending = false;
270
271         RequestFileChunk();
272       }
273
274       private void RequestFileChunk()
275       {
276         if (fileStream.Length < length)
277         {
278           var stream = new MemoryStream();
279           var writer = new BinaryWriter(stream);
280
281           writer.Write((byte)MessageType.FileRequest);
282           writer.Write(fileStream.Length);
283           SendPacket(stream.ToArray());
284         }
```

```
285         else
286         {
287           fileStream.Close();
288           fileStream = File.OpenRead(tempPath);
289           var hash = new SHA512CryptoServiceProvider().ComputeHash(fileStream);
290           fileStream.Close();
291           var myHashName = "";
292           foreach (var b in hash)
293           {
294             myHashName += b.ToString("x2");
295           }
296           if (hashName == myHashName)
297           {
298             File.Delete(filePath);
299             File.Move(tempPath, filePath);
300             SendPacket(new byte[] { (byte)MessageType.FileReceived });
301           }
302           else
303           {
304             File.Delete(tempPath);
305             fileStream = File.OpenWrite(tempPath);
306             RequestFileChunk();
307           }
308         }
309       }
310
311       private void SendFileChunk(BinaryReader reader)
312       {
313         if (sending != true)
314         {
315           return;
316         }
317
318         timer.Stop();
319
320         DisconnectButton.IsEnabled = true;
321         StatusText.Text = "Логическое соединение установлено.";
322
323         var allLength = fileStream.Length;
324         var position = reader.ReadInt64();
325         ProgressBar.Value = (double)position / allLength;
326
327         var stream = new MemoryStream();
328         var writer = new BinaryWriter(stream);
329
330         writer.Write((byte)MessageType.FileChunk);
331
332         var remaining = fileStream.Length - position;
333         if (remaining < chunkSize)
334         {
```

```
335           writer.Write(position);
336           writer.Write((short)remaining);
337           var buffer = new byte[remaining];
338           fileStream.Read(buffer, 0, (int)remaining);
339           writer.Write(buffer);
340         }
341         else
342         {
343           writer.Write(position);
344           writer.Write(chunkSize);
345           var buffer = new byte[chunkSize];
346           fileStream.Read(buffer, 0, chunkSize);
347           writer.Write(buffer);
348         }
349
350         SendPacket(stream.ToArray());
351       }
352
353       private void SaveFileChunk(BinaryReader reader)
354       {
355         if (sending != false)
356         {
357           return;
358         }
359
360         timer.Stop();
361
362         var position = reader.ReadInt64();
363         var chunk = reader.ReadInt16();
364         var buffer = reader.ReadBytes(chunk);
365
366         fileStream.Seek(position, SeekOrigin.Begin);
367         fileStream.Write(buffer, 0, chunk);
368
369         ProgressBar.Value = (double)(position + chunk) / length;
370
371         RequestFileChunk();
372       }
373
374       private void SendPacket(byte[] packet)
375       {
376         lastPacket = packet;
377         retries = 0;
378         DataLink.SendPacket(packet);
379         timer.Start();
380       }
381
382       private void ResendPacket(object sender, EventArgs e)
383       {
384         if (retries++ < maxRetries)
```

```
385         {
386           DataLink.SendPacket(lastPacket);
387           return;
388         }
389
390         if (lastPacket[0] == (byte)MessageType.FileReceivedOk ||
391           lastPacket[0] == (byte)MessageType.FileReceived ||
392           lastPacket[0] == (byte)MessageType.Disconnect ||
393           lastPacket[0] == (byte)MessageType.DisconnectOk)
394         {
395           timer.Stop();
396           return;
397         }
398
399         CloseButton.IsEnabled = true;
400         FileBox.IsEnabled = true;
401         DirectoryBox.IsEnabled = true;
402         DisconnectButton.IsEnabled = true;
403         if (fileStream != null)
404         {
405           fileStream.Close();
406         }
407
408         if (sending == true && fileStream.Position == 0)
409         {
410           StatusText.Text = "Физическое соединение открыто.";
411           MessageBox.Show(
412             "Принимающая сторона не готова к логическому соединению.");
413         }
414         else
415         {
416           SendPacket(new byte[] { (byte)MessageType.Disconnect });
417           StatusText.Text = "Логическое соединение потеряно.";
418           MessageBox.Show("Логическое соединение потеряно.");
419         }
420         sending = null;
421       }
422
423       private void PortCheck(bool DSR, bool CTS, bool DTR, bool RTS)
424       {
425         Dispatcher.Invoke(new Physical.PortListener(RealPortCheck),
426           DispatcherPriority.Send, DSR, CTS, DTR, RTS);
427       }
428
429       private void RealPortCheck(bool DSR, bool CTS, bool DTR, bool RTS)
430       {
431         DtrIndicator.IsChecked = DTR;
432         DsrIndicator.IsChecked = DSR;
433         RtsIndicator.IsChecked = RTS;
434         CtsIndicator.IsChecked = CTS;
```

```
435       }
436
437       private void ExceptionHandler(object sender,
438         DispatcherUnhandledExceptionEventArgs e)
439       {
440         e.Handled = true;
441         MessageBox.Show(e.Exception.Message);
442       }
443
444       public void Dispose()
445       {
446         fileDialog.Dispose();
447         folderDialog.Dispose();
448       }
449
450       private enum MessageType:byte
451       {
452         FileName, ReceiveNotReady, FileRequest, FileChunk,
453         FileReceived, FileReceivedOk, Disconnect, DisconnectOk
454       }
455     }
456   }
```

5. Файл Network.cs

```
1   using System;
2   using System.Collections;
3   using System.Collections.Generic;
4   using System.Linq;
5   using System.IO.Ports;
6   using System.Timers;
7
8   namespace iu5nt
9   {
10    public static class DataLink
11    {
12      static byte[] currentPacket;
13      static byte[] checkSumm;
14      static int length = 0;
15      static List<byte> recievedPacket = new List<byte>();
16      static List<bool> recievedPacketBuffer = new List<bool>();
17      static List<bool> debugBuffer = new List<bool>();
18      static bool firstTrigger = false;
19      static bool secondTrigger = false;
20      static bool screenTrigger = false;
21      static int firstTPosition = 0;
22      static Timer cleanerTimer = new Timer(1000);
23      public delegate void RecieveMEthod(byte[] packet, bool check);
24      public static event RecieveMEthod OnRecieve;
25
```

```
26      static DataLink(){
27        cleanerTimer.Elapsed += new ElapsedEventHandler(TimerListener);
28      }
29      static void TimerListener(object sender, ElapsedEventArgs e){
30        if(firstTrigger){
31          recievedPacket.Clear();
32          recievedPacketBuffer.Clear();
33          firstTrigger = false;
34          secondTrigger = false;
35          firstTPosition = 0;
36        }
37
38      }
39      public static void RecievePacket(BitArray recievedBit)
40      {
41        bool[] bbuffer = new bool[11];
42        recievedBit.CopyTo(bbuffer, 0);
43        recievedPacketBuffer.AddRange(bbuffer);
44        debugBuffer.AddRange(bbuffer);
45        while (recievedPacketBuffer.Count >= 8)
46        {
47
48          bool[] seriousBuffer = recievedPacketBuffer.GetRange(0, 8).ToArray();
49          recievedPacketBuffer.RemoveRange(0, 8);
50          var bitBufff = new BitArray(seriousBuffer);
51          byte[] recieved = new byte[1];
52          bitBufff.CopyTo(recieved, 0);
53          if (firstTrigger || recieved[0] == 0xFF) {
54            if(!firstTrigger){
55              cleanerTimer.Start();
56            }
57            recievedPacket.AddRange(recieved);
58            firstTrigger = true;
59          }
60
61          var packLen = recievedPacket.Count;
62          if (packLen > 6)
63          {
64            for (var k = 1; k > 0 && packLen > 3; k--)
65            {
66              if (recievedPacket[packLen - k] == 0xFF &&
67                recievedPacket[packLen - k - 1] == 0xFE && !screenTrigger)
68              {
69                recievedPacket.RemoveAt(packLen - k - 1);
70                packLen--;
71                screenTrigger = true;
72              }
73              else
74              {
75                if (recievedPacket[packLen - k] == 0xFF)
```

```
76                {
77                  if (!secondTrigger)
78                  {
79                    secondTrigger = true;
80                    firstTPosition = packLen - k - 1;
81                  }
82                }
83                else
84                {
85                  if (recievedPacket[packLen - k] == 0xFE &&
86                    recievedPacket[packLen - k - 1] == 0xFE && !screenTrigger)
87                  {
88                    recievedPacket.RemoveAt(packLen - k - 1);
89                    packLen--;
90                    screenTrigger = true;
91                  } else {
92                    if (screenTrigger){
93                      screenTrigger = false;
94                    }
95                  }
96                }
97
98              }
99            }
100
101         }
102       }
103       if (secondTrigger)
104       {
105         var exactPacket = recievedPacket.Skip(5)
106                           .Take(firstTPosition - 4).ToArray();
107         var buffer = 0;
108         foreach (var packByte in exactPacket)
109         {
110           buffer += packByte;
111         }
112         //Тут может быть ошибка по длине для проверки суммы
113         var checksummP = recievedPacket.Skip(1).Take(4).ToArray();
114         if (buffer == BitConverter.ToUInt32(checksummP, 0))
115         {
116           OnRecieve(exactPacket, true);
117         } else
118         {
119           OnRecieve(exactPacket, false);
120         }
121         recievedPacket.Clear();
122         recievedPacketBuffer.Clear();
123         firstTrigger = false;
124         secondTrigger = false;
125         firstTPosition = 0;
```

```
126           cleanerTimer.Stop();
127         }
128       }
129       public static void SendPacket(byte[] newPacket)
130       {
131         recievedPacket.Clear();
132         firstTrigger = false;
133         secondTrigger = false;
134         firstTPosition = 0;
135         recievedPacketBuffer.Clear();
136         length = 0;
137         currentPacket = newPacket;
138         length += currentPacket.Length;
139         var summBuffer = 0;
140         foreach (var item in currentPacket)
141         {
142           summBuffer += item;
143         }
144         checkSumm = BitConverter.GetBytes(summBuffer);
145         length += checkSumm.Length;
146         var indexPacket = currentPacket
147           .SelectMany(x => (x == 0xFE || x == 0xFF) ?
148             new byte[] { 0xFE, x } :
149             new byte[] { x })
150           .ToList();
151         indexPacket.Add(0xFF);
152         List<byte> indexSumm = new List<byte>(checkSumm);
153         indexSumm.AddRange(indexPacket);
154         indexSumm.Insert(0,0xFF);
155         BitArray bitPacket = new BitArray(indexSumm.ToArray());
156         Physical.Send(bitPacket);
157       }
158
159     }
160
161
162     public static class Physical
163     {
164       static SerialPort _serialPort;
165       public static bool connected = false;
166       public static List<UInt32> failList = Learning();
167       public delegate void PortListener(bool DSR, bool CTS, bool DTR, bool RTS);
168       public static event PortListener OnCheck;
169       public static event PortListener UICheck;
170       public static void SetRts(bool setter)
171       {
172         _serialPort.RtsEnable = setter;
173         OnCheck(_serialPort.DsrHolding, _serialPort.CtsHolding,
174                 _serialPort.DtrEnable, _serialPort.RtsEnable);
175       }
```

```
176       static void StatusCheck (Object sender, SerialPinChangedEventArgs e) {
177         UICheck(_serialPort.DsrHolding, _serialPort.CtsHolding,
178                 _serialPort.DtrEnable, _serialPort.RtsEnable);
179       }
180       public static void Connect(String portName)
181       {
182         if (connected)
183         {
184           Disconnect();
185         }
186         _serialPort = new SerialPort(portName)
187         {
188           BaudRate = 115200,
189           DtrEnable = true,
190           ReceivedBytesThreshold = 2
191         };
192         _serialPort.DataReceived +=
193           new SerialDataReceivedEventHandler(DataReceivedHandler);
194         _serialPort.PinChanged += new SerialPinChangedEventHandler(StatusCheck);
195         _serialPort.Open();
196         OnCheck(_serialPort.DsrHolding, _serialPort.CtsHolding,
197                 _serialPort.DtrEnable, _serialPort.RtsEnable);
198         connected = true;
199       }
200       public static void Disconnect()
201       {
202         _serialPort.Close();
203         connected = false;
204       }
205       private static void DataReceivedHandler(object sender,
206         SerialDataReceivedEventArgs e)
207       {
208         var i = 0;
209         while(_serialPort.BytesToRead > 1)
210         {
211           i++;
212           var byteBuffer = new byte[4];
213           _serialPort.Read(byteBuffer,0,2);
214           var dec = DeCycle(byteBuffer);
215           DataLink.RecievePacket(dec);
216         }
217       }
218       static public List<UInt32> Learning()
219       {
220         UInt32 study = 16384;
221         UInt32 qbite = study;
222         UInt32 osn = 19;
223         while (qbite > 15)
224         {
225           UInt32 clone = osn;
```

```
226           clone = clone << ((int)Math.Log(qbite, 2) - 4);
227           qbite ^= clone;
228         }
229         study = study + qbite;
230         var getBuffed = new List<UInt32>();
231         for (var i = 0; i < 15; i++)
232         {
233           var bits = BitConverter.GetBytes(study);
234           var beef = new BitArray(new byte[] { bits[0],bits[1]});
235           beef.Set(i, !beef.Get(i));
236           var number = new Int32[1];
237           beef.CopyTo(number, 0);
238           qbite = (uint)number[0];
239           while (qbite > 15)
240           {
241             var clone = osn;
242             clone = clone << ((int)Math.Log(qbite, 2) - 4);
243             qbite ^= clone;
244           }
245           getBuffed.Add(qbite);
246         }
247         return getBuffed;
248       }
249       static BitArray DeCycle(byte[] cycled)
250       {
251
252         var buffer = BitConverter.ToUInt32(cycled,0);
253         UInt32 qbite = buffer;
254         UInt32 osn = 19;
255         while (qbite > 15)
256         {
257           UInt32 clone = osn;
258           clone = clone << ((int)Math.Log(qbite, 2) - 4);
259           qbite ^= clone;
260         }
261         if (qbite == 0)
262         {
263           var convert = BitConverter.GetBytes(buffer / 16);
264           var bitar = new BitArray(convert)
265           {
266             Length = 11
267           };
268           return bitar;
269         }
270         else
271         {
272           var indi = failList.IndexOf(qbite);
273           if(indi > 3)
274           {
275             var convert = BitConverter.GetBytes(buffer / 16);
```

```
276             var bitar = new BitArray(convert)
277             {
278               Length = 11
279             };
280             bitar.Set(indi - 4, !bitar.Get(indi - 4));
281             return bitar;
282           } else
283           {
284             var convert = BitConverter.GetBytes(buffer / 16);
285             var bitar = new BitArray(convert)
286             {
287               Length = 11
288             };
289             return bitar;
290           }
291
292         }
293       }
294       public static void Send(BitArray allBits)
295       {
296         var work = new bool[allBits.Count];
297         allBits.CopyTo(work, 0);
298         int iterate = (allBits.Count - 1) / 11 + 1;
299         for (var i = 0; i < iterate; i++)
300         {
301           var array = work.Skip(i * 11).Take(11).ToArray();
302           if(_serialPort.CtsHolding && _serialPort.DsrHolding){
303             _serialPort.Write(GetCycled(new BitArray(array)), 0, 2);
304           } else {
305             OnCheck(_serialPort.DsrHolding, _serialPort.CtsHolding,
306                     _serialPort.DtrEnable, _serialPort.RtsEnable);
307             return;
308           }
309         }
310       }
311       private static byte[] BitArrayToByteArray(BitArray bits)
312       {
313         byte[] ret = new byte[(bits.Length - 1) / 8 + 1];
314         bits.CopyTo(ret, 0);
315         return ret;
316       }
317       private static byte[] GetCycled(BitArray eleven) {
318         var buffer = new Int32[1];
319         eleven.CopyTo(buffer, 0);
320         UInt32 qbite = (uint)buffer[0];
321         qbite *= 16;
322         UInt32 osn = 19;
323         while (qbite > 15)
324         {
325           UInt32 clone = osn;
```

```
326           clone = clone << ((int)Math.Log(qbite, 2) - 4);
327           qbite ^= clone;
328         }
329         byte[] result = BitConverter.GetBytes(buffer[0] * 16 + qbite);
330         return new byte[] { result[0], result[1] };
331 
332       }
333     }
334   }
```