

3.1 Пространство имен System.Drawing

Графический интерфейс приложений C# (GDI+), как и других приложений, предназначенных для работы в рамках Microsoft .NET Framework, состоит из набора классов, объединяемых пространством имен. Одно из основных пространств имен GDI+ языка C# является пространство имен System.Drawing. Классы этого пространства имен определяют перечень объектов и инструментов, предназначенных для «рисования».

К наиболее часто используемым классам пространства имен System.Drawing относятся:

Brush (Brushes, SolidBrush и др.). Объекты Brush (кисть) используются для заполнения пространства внутри геометрических фигур. Тип Brush — это абстрактный базовый класс, остальные типы являются производными от Brush и определяют разные наборы возможностей.

Pen (Pens, SystemPens). Pen (перо) — это объект класса, при помощи которого можно рисовать прямые и кривые линии. В классе Pen определен набор статических свойств, при помощи которых можно получить объект Pen с заданными свойствами (например, с установленным цветом)

Font (FontFamily). Объекты типа Font определяют характеристики шрифта (имя, размер, начертание и т. п.). FontFamily представляет набор шрифтов, которые относятся к одному семейству, но имеют некоторые небольшие отличия.

Graphics. Этот класс определяет набор свойств и методов для вывода текста, изображений и геометрических фигур на экран монитора. Он позволяет приложению работать с контекстом устройств системы Windows.

Region. Этот класс определяет область, занятую геометрической фигурой.

Point (PointF). Эти структуры обеспечивают работу с координатами точки. Point работает со значениями типа int, а PointF — со значениями типа float.

В пространстве имен System.Drawing также находятся классы Icon, Image, Color, Bitmap и другие классы так или иначе связанные с отображением графической информации на экране монитора.

3.2 Класс Graphics

Основным классом для «рисования» в языке C# является класс Graphics. Он предназначен для вывода графической информации в клиентскую часть формы приложения. Для того чтобы приложение могло что-нибудь нарисовать в окне, оно должно получить или создать для этого окна объект класса Graphics. Далее, пользуясь свойствами и методами этого объекта, приложение может рисовать в окне различные фигуры или текстовые строки.

Прежде чем создавать в приложении объект класса Graphics необходимо определиться с обработчиком события по «рисованию». В системе Windows за перемещением и изменением размера окон «следит» специальное сообщение WM_PAINT, которое при необходимости извещает приложения, о том, что им следует перерисовать содержимое окна. Любые действия с окном – перемещение его по рабочему столу экрана монитора, изменение его размеров и т.д. сопровождается требованием системы Windows «перерисовать» окно. В приложении обработчик события WM_PAINT, получив такое сообщение, должен выполнить перерисовку всего окна или его части, в зависимости от дополнительных данных, полученных вместе с сообщением WM_PAINT. Для создания заготовки обработчика сообщения WM_PAINT формы необходимо в окне свойств окна формы дважды кликнуть мышкой на пункт PAINT (смотри рисунок 3.1).

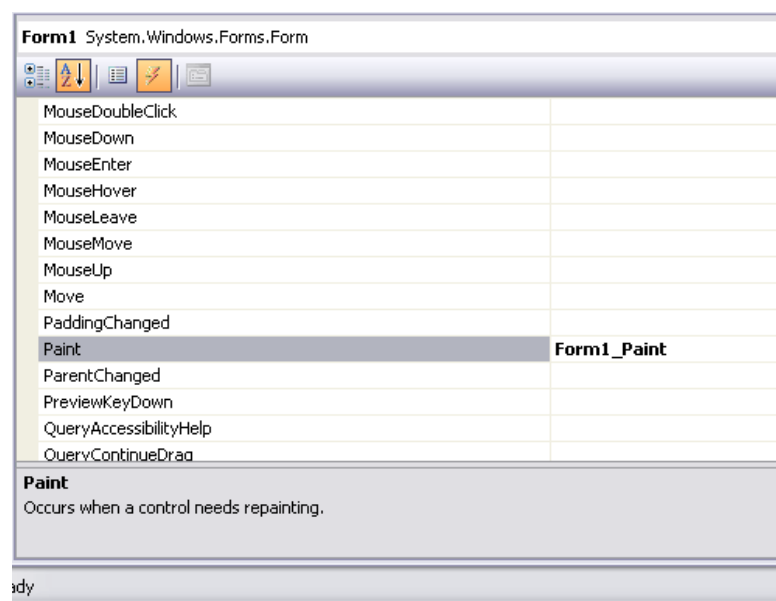


Рисунок 3.1 – Создание обработчика WM_PAINT

В результате будет создан обработчик события WM_PAINT (точнее Form1_Paint – как это видно на рисунке 3.1). Этот обработчик будет получать управление всякий раз, когда по тем или иным причинам возникнет необходимость в перерисовке содержимого окна нашего приложения.

Вот в каком виде будет создан обработчик события Paint:

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
}
```

Обработчику Form1_Paint передаются два параметра.

Через первый параметр передается ссылка на объект, вызвавший событие. В нашем случае это будет ссылка на форму Form1 (где рисовать).

Что же касается второго параметра, то через него передается ссылка на объект класса PaintEventArgs. Этот объект имеет свойство ClipRectangle, доступное только для чтения. Через свойство ClipRectangle передаются

границы области, которую должен перерисовать обработчик события Paint. Эти границы передаются в виде объекта класса Rectangle. Свойства этого класса Left, Right, Width и Height, наряду с другими свойствами, позволяют определить расположение и размеры области. По умолчанию обработчик события Paint игнорирует свойство ClipRectangle, перерисовывая содержимое окна полностью. Например,

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
    Pen myPen = new Pen(Color.Red, 2);
    Graphics g = e.Graphics;
    //g.DrawEllipse(myPen, x, y, ra, ra);
    g.DrawEllipse(myPen, 100, 100, 100, 100);
}
```

В данном примере создается «перо» для рисования красным цветом и толщиной 2 пикселя – объект myPen. Создается объект g типа Graphics для перерисовываемой области (по умолчанию вся форма). Обратите внимание нет new. Далее для объекта g запускается метод рисования эллипса.

Особенность объекта g типа Graphics заключается в том, что фактически объект это указатель на контекст устройства монитора (специальные программы системы Windows, связывающие приложение с драйвером видеокарты компьютера). С помощью контекстов устройств система Windows обеспечивает совместимость приложений и драйверов устройств компьютера, например, независимо от типа видеокарты код нашего приложения останется неизменным, а все проблемы управления видеокартой решает контекст устройства монитора.

Работа нашей программы приведена на рисунке 3.2

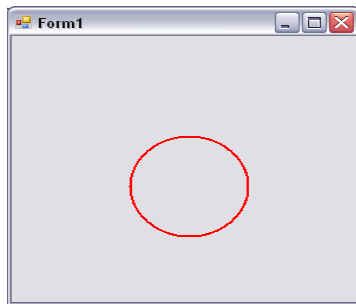


Рисунок 3.2 – Отображение эллипса

Как и большинство классов C# класс Graphics имеет свойства и методы. Рассмотрим некоторые из них.

Clear() – этот метод заполняет объект Graphics выбранным пользователем цветом, удаляя его предыдущее содержимое.

Большая группа методов для «рисования» некоторых геометрических фигур: DrawArc(), DrawBezier(), DrawBeziers(), DrawCurve(), DrawEllipse(), DrawIcon(), DrawLine(), DrawLines(), DrawPie(), DrawPath(), DrawRectangle(), DrawRectangles(), DrawString().

Для «заполнения» внутренних областей геометрических фигур используются методы перед которыми находится слово Fill, например, FillPie(), FillEllipse() или FillRectangle().

Работу некоторых методов можно изучить в книге «Визуальное проектирование приложений C#» авторов А.В. Фролов, Г.В. Фролов (глава 10 – смотри перечень рекомендуемой литературы). Необходимо отметить, что получение объекта типа Graphics, для перерисовываемых областей возможно не только с помощью объекта класса PaintEventArgs обработчика событий Form1_Paint. Можно использовать метод CreateGraphics, описанного в классах формы и элемента управления, например,

```
Graphics g = this.CreateGraphics(); или  
Graphics g = Graphics.FromHwnd(this.Handle);
```

Можно создать графический объект с помощью объекта-потомка Image. Этот способ используется для изменения существующего изображения, например,

```
Bitmap bm = new Bitmap( "d:\\picture.bmp" );  
Graphics g = Graphics.FromImage( bm );
```

3.3 Пример программной реализации

В качестве учебной программы рассмотрим вывод на экран монитора структуры типа граф и некоторых стандартных графических фигур.

Код программы:

```
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Linq;  
using System.Text;  
using System.Windows.Forms;  
  
namespace WindowsFormsApplication1  
{  
    public partial class Form1 : Form  
    {  
        public Form1()  
        {  
            InitializeComponent();  
        }  
        public static int p = 0;  
        private void Form1_Paint(object sender, PaintEventArgs e)  
        {  
            Graphics g = e.Graphics;  
            if (p == 0)  
            {  
                String st;  
                int i, j;  
                int xc, yc;
```

```

//g.Clear(Color.White);
int[,] a = new int[9, 9]
    {{0, 6, 1000, 4, 1000, 1000, 1000, 1000, 1000},
     {6, 0, 5, 1000, 3, 6, 1000, 1000, 1000},
     {1000, 5, 0, 5, 1000, 3, 1000, 1000, 1000},
     {4, 1000, 5, 0, 1000, 1000, 4, 1000, 1000},
     {1000, 3, 1000, 1000, 0, 7, 1000, 7, 1000},
     {1000, 6, 3, 1000, 7, 0, 6, 4, 1000},
     {1000, 1000, 1000, 4, 1000, 6, 0, 9, 6},
     {1000, 1000, 1000, 1000, 7, 4, 9, 0, 8},
     {1000, 1000, 1000, 1000, 1000, 1000, 6, 8, 0}};

int[,] V = new int[9, 2]
    {{200, 40},
     {100, 80},
     {200, 100},
     {280, 60},
     { 60, 140},
     {200, 160},
     {340, 140},
     {160, 220},
     {360, 260}};

e.Graphics.Clear(Color.Bisque);
Pen myPen = new Pen(Color.Red, 2);
for (i = 0; i < 9; i++)
    for (j = 0; j < 9; j++)
        if ((a[i, j] != 0) && (a[i, j] != 1000))
        {
            g.DrawLine(myPen, V[i, 0], V[i, 1], V[j, 0], V[j, 1]);
            xc = (int)(V[i, 0] + V[j, 0]) / 2;
            yc = (int)(V[i, 1] + V[j, 1]) / 2;
            st = Convert.ToString(a[i, j]);
            g.DrawString(st, new Font("Times new Roman", 8),
Brushes.Blue, xc - 6, yc - 15);
        }
    for (i = 0; i < 9; i++)
    {
        g.FillEllipse(Brushes.White, V[i, 0] - 12, V[i, 1] - 12, 25,
25);
        st = Convert.ToString(i);
        g.DrawString(st, new Font("10_IC_1", 12), Brushes.Black,
V[i, 0] - 6, V[i, 1] - 6);
    }
}
if (p == 1)
{
    g.Clear(Color.White);
    g.DrawRectangle(new Pen(Brushes.Blue, 2), 5, 5, 380, 270);

    Pen myPen = new Pen(Color.Black, 2);
    Point[] myPoints =
    {
        new Point(10, 10),

```

```

        new Point(100, 40),
        new Point(150, 24),
        new Point(100, 100),
    };
    g.DrawPolygon(myPen, myPoints);

    g.DrawLine(myPen, 10, 250, 360, 250);
    g.DrawLine(myPen, 10, 250, 10, 150);
    g.DrawString("Y", new Font("10_IC_1", 12), Brushes.Black,
5,130);
    g.DrawString("X", new Font("10_BT_1", 12), Brushes.Black,
360,240);
    g.FillRectangle(Brushes.Blue, 25, 180, 25, 70);
    g.FillRectangle(Brushes.Red, 75, 150, 25, 100);
    g.FillRectangle(Brushes.Green, 125, 200, 25, 50);

    g.FillPie(Brushes.Blue, 170, 10, 200, 150, 0, 120);
    g.FillPie(Brushes.Red, 170, 10, 200, 150, 120, 120);
    g.FillPie(Brushes.Green, 170, 10, 200, 150, 240, 120);
}
}
private void button2_Click(object sender, EventArgs e)
{
    if (p == 0) p = 1; else p = 0;
    this.Invalidate();
}
}
}

```

Работа программы

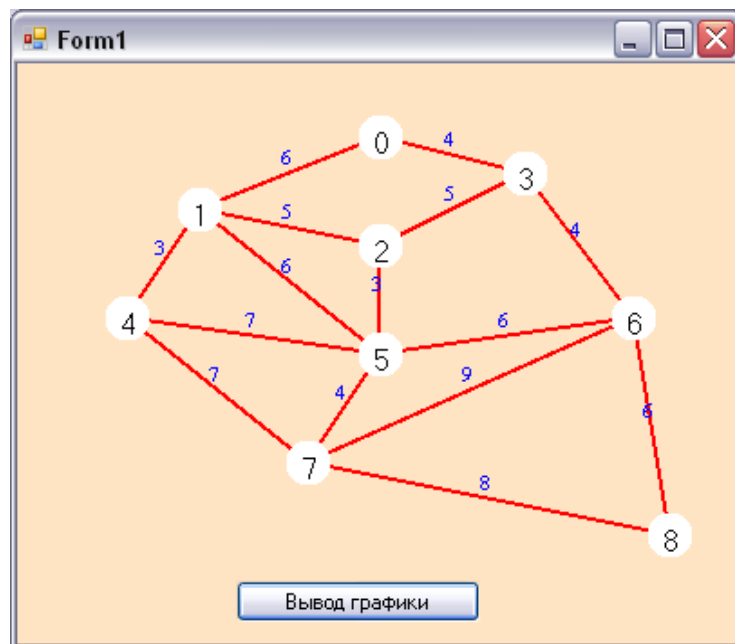


Рисунок 3.3 – Вывод графа

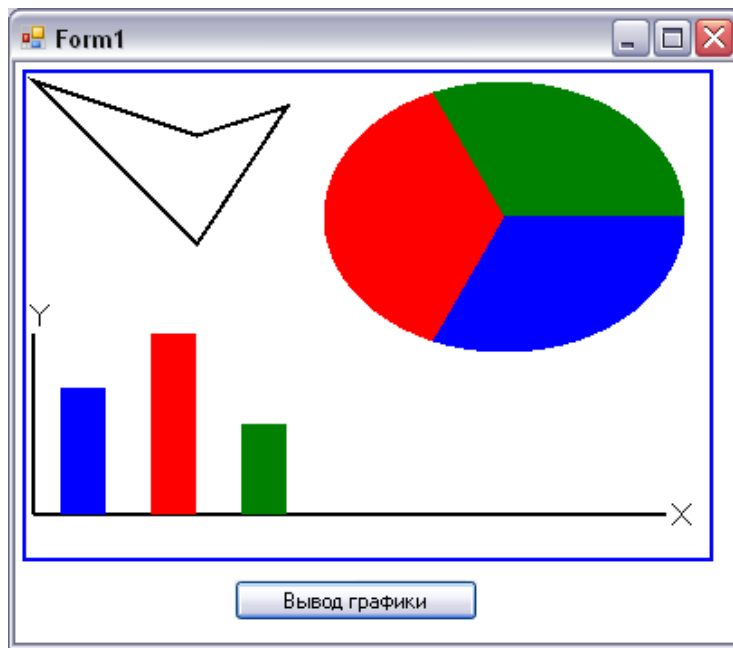


Рисунок 3.4 – Вывод различных графических фигур

Базовой литературой этого примера является книга «Визуальное проектирование приложений C#» авторов А.В. Фролов, Г.В. Фролов (глава 10)

В программе управление выводом информации осуществляется с помощью глобальной переменной p . Если $p = 0$ (при запуске программы), то в окно формы выводится рисунок графа. При нажатии кнопки «Вывод графики» в окне формы значение p меняется на противоположное – если p было равно 0, то оно станет равным 1 и наоборот.

В обработчике события нажатия кнопки присутствует метод, применяемый для текущего объекта – `this.Invalidate()`; (указатель `this` всегда содержит адрес текущего объекта – объекта, с которым работает программа). В нашем примере текущим объектом является форма. Метод `this.Invalidate()`; требует от операционной системы Windows сформировать для формы сообщение `WM_PAINT`. Получив это сообщение, наша программа запустит обработчик события `private void Form1_Paint(object sender, PaintEventArgs e)`, т.е. перерисует содержимое окна формы при новых значениях переменной p . Таким образом, можно из приложения «формировать» сообщение `WM_PAINT` операционной системы Windows.

Рассмотрим подробнее код учебной программы. Первая часть содержит рисунок графа из предыдущего семестра – заимствована матрица смежности. Дополнительно введен массив координат вершин графа – V . Предварительно граф был нарисован на листе бумаги и значения координат (в масштабе) взяты непосредственно с рисунка. Очищаем окно формы `e.Graphics.Clear(Color.Bisque);` – закрашиваем его цветом `Color.Bisque`. Устанавливаем значение цвет «пера» красный и толщину линий 2 пикселя – `Pen myPen = new Pen(Color.Red, 2);`. Запускаем

циклы рисования линий между вершинами графа. Одновременно вычисляются координаты x_c и y_c для средней точки между вершинами графа. Используя эти координаты, мы над линией ребра выводим его значение, взятое из матрицы смежности. При выводе текста по формату записи `g.DrawString` необходимо указывать тип шрифта и его размер. В следующем цикле рисуются и подписываются вершины графа. Метод `g.FillEllipse` рисует «закрашенный» эллипс, вписанный в прямоугольную область, расположение и размеры которой передаются ему в качестве параметров.

На «втором» окне формы выполнена «очистка» окна белым цветом. В учебных целях «нарисованы» многоугольник, линии осей координат X и Y , набор прямоугольников (гистограмма) и круговая диаграмма.

Метод `g.DrawPolygon(myPen, myPoints);` позволяет нарисовать многоугольник, координат вершин которого передаются массивом. Первый параметр метода определяет цвет линии, которой рисуется многоугольник.

Метод `g.FillRectangle` позволяет рисовать закрашенные прямоугольники, заданные координатой верхнего левого угла, а также шириной и высотой. В качестве первого параметра этим методам передается цвет «заполнения» внутренней области прямоугольника.

Круговая диаграмма рисуется с помощью метода `g.FillPie` предназначенного для отображения «закрашенных» сегментов. В качестве первого параметра методу нужно передать цвет «заполнения» внутренней области сегмента. Следующие четыре параметра задают расположение и размеры прямоугольника, в который вписывается эллипс. Последние два параметра определяют углы, ограничивающие сегмент эллипса – начальный угол и его приращение по часовой стрелке.

3.4 Отображение информации в текстовом и графическом виде.

Для отображения информации в текстовом виде мы будем использовать элемент управления `TextBox`, а отображение информации в графическом виде (подобие экрана осциллографа) – элемент `Chart`.

Работу этих элементов рассмотрим на примере учебной задачи – отобразить в окне программы «временные диаграммы» работы трехразрядного суммирующего двоичного счетчика. Переключение счетчика осуществлять тактирующими импульсами, формируемыми по прерыванию от элемента `Timer`. Запуск и остановку таймера будем осуществлять одной кнопкой «Пуск – Стоп». Элемент таймер не используется в наших программах обмена данными ПК и МК, но он избавляет нас от многократного нажимания на кнопку при имитации тактирующих импульсов.

Проект разработан в `WindowsFormsApplication`. С помощью мастера визуального программирования и панели `Toolbox` разместим на форме проекта необходимые элементы управления (рисунок 3.5).

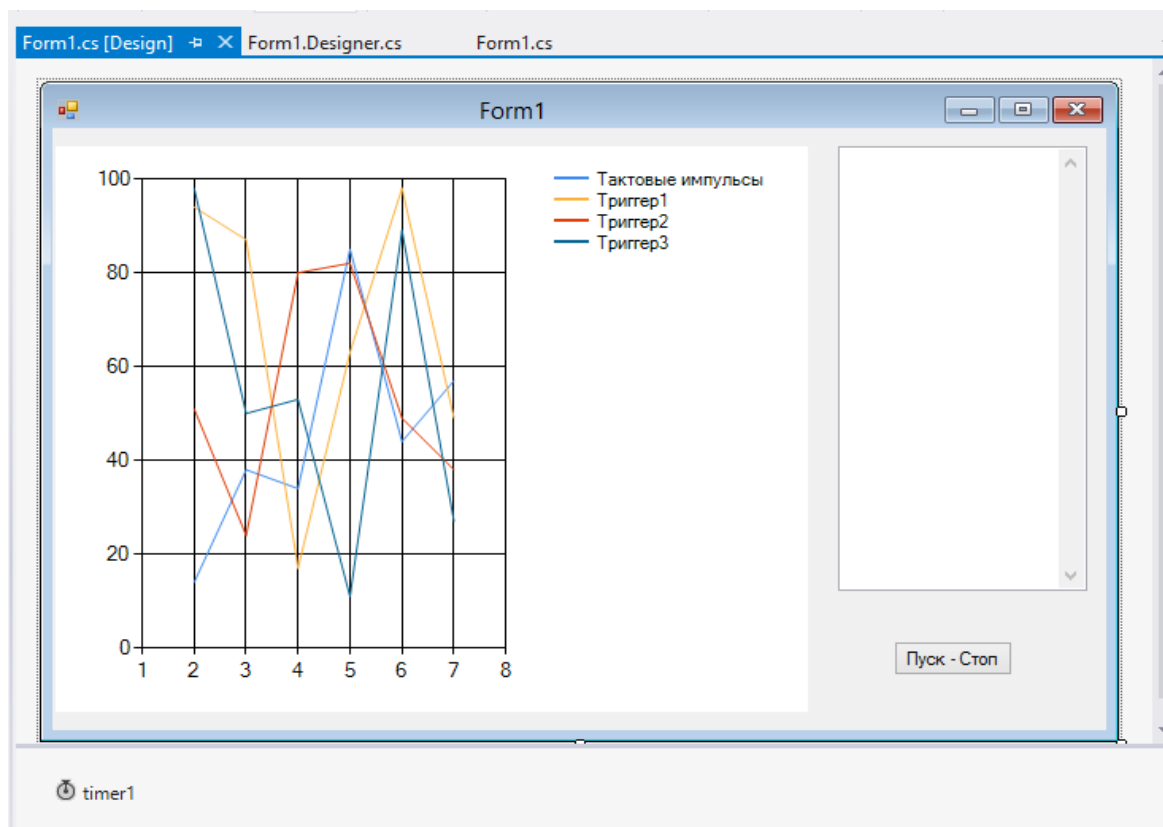


Рисунок 3.5 - Размещение элементов управления на форме проекта

В свойствах элемента chart1 (рисунок 3.6) редактируем позицию Series в соответствии с рисунком 3.7.

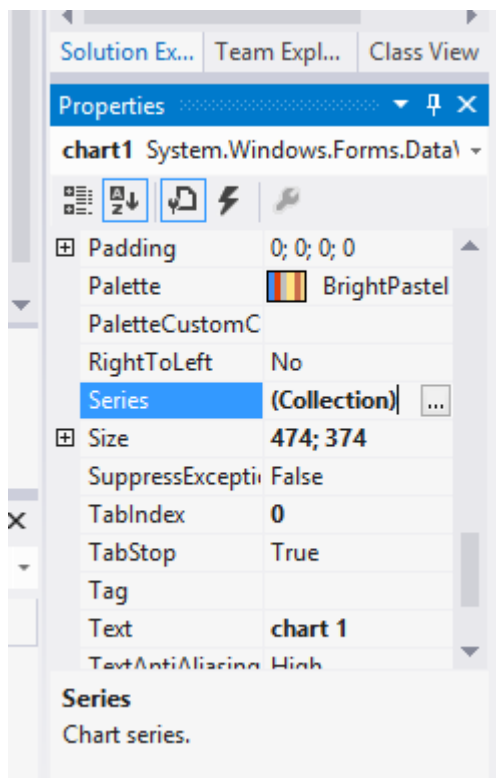


Рисунок 3.6 - Свойства элемента управления Chart1

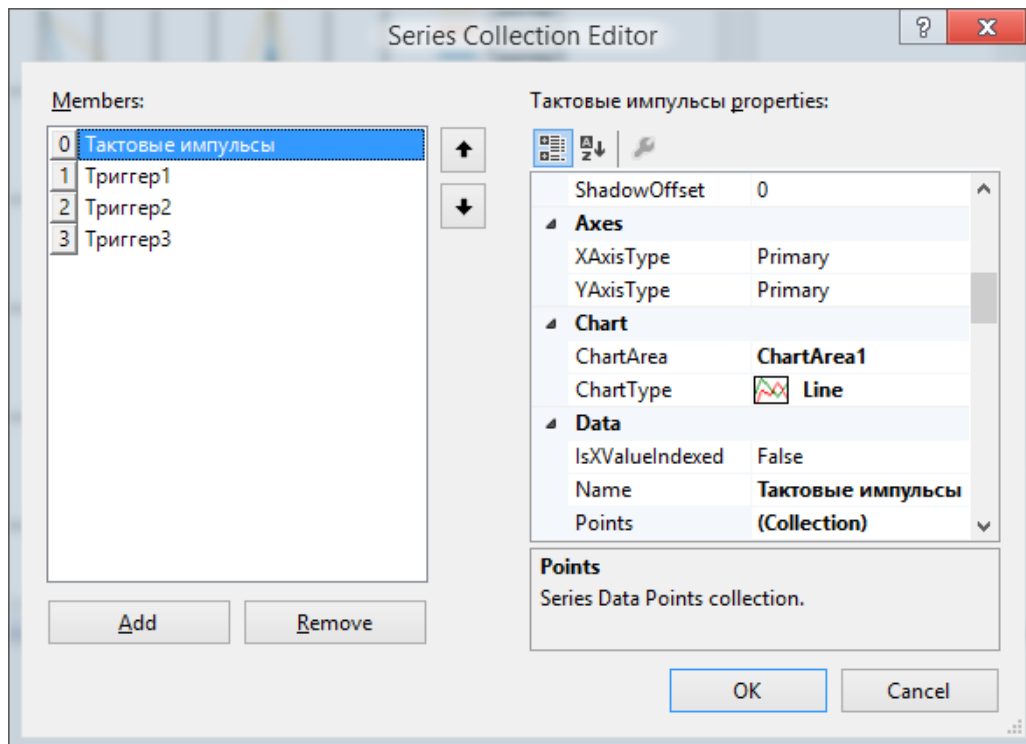


Рисунок 3.7 - Окно редактирования Series элемента Chart1

Для элемента timer1 в его свойствах на страничке Events устанавливаем прерывание timer1_Tick, а на общей страничке свойств задаем свойству Interval значение 1000 - время прерывания 1 секунда. В результате работы со свойствами элемента timer1 в коде формы1 появится "пустая заготовка" на прерывания от таймера:

```
private void timer1_Tick(object sender, EventArgs e)
{
}
```

Для элемента Button1 в его свойствах редактируем значение свойства Text - устанавливаем «Пуск – Стоп». Двойным нажатием левой кнопки мышки на изображение кнопки на форме создадим "пустой обработчик", в который занесем код управления таймером:

```
private void button1_Click(object sender, EventArgs e)
{
    if (timer1.Enabled == false) timer1.Enabled = true;
    else timer1.Enabled = false;
}
```

Разместим в окне формы элемент управления textBox1, для которого зададим значение свойства Multiline равное True. Остальные значения свойств оставляем без изменений.

На этом этап визуального программирования закончен и Мы переходим к этапу написания кода программы.

Для улучшения восприятия кода программы напишем два метода AddPoints и AddText.

В методе AddPoints Мы опишем работу элемента chart1 при формировании очередных "точек" графиков временных диаграмм.

В методе AddText Мы будем формировать значения строковой переменной ss для отображения очередного состояния счетчика в текстовом виде в окне элемента textBox1. Состояние счетчика будем отображать в традиционном табличном виде с помощью нулей и единиц.

В обработчике прерывания таймера реализован алгоритм сложения четырехразрядных "двоичных чисел". Для "независимого" отображения графика каждого разряда "двоичного числа" использованы различные начальные нулевые значения:

```
static public byte[] tim = new byte[4] { 1, 6, 11, 16 };
```

Без учета начальных нулевых значений данные в каждом разряде могут изменяться от 1 - "нулевое" значение, до 4 - "единичное" значение.

Для предания тактовым "импульсам" формы прямоугольников использован цикл for с пятикратным повтором вывода данных в элемент chart1.

При инициализации формы в элементе chart1 формируются "нулевые" значения, а для textBox1 выводится заголовок таблицы.

Исходный код файла Form1.cs:

```
using System;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        private void AddPoints(byte[] points)
        {
            chart1.Series[0].Points.AddY(points[0]);
            chart1.Series[1].Points.AddY(points[1]);
            chart1.Series[2].Points.AddY(points[2]);
            chart1.Series[3].Points.AddY(points[3]);
        }

        private void AddText(byte[] points)
        {
            string ss = "";
            if (points[3] == 16) ss = ss + "  0"; else ss = ss + "  1";
            if (points[2] == 11) ss = ss + "  0"; else ss = ss + "  1";
            if (points[1] == 6) ss = ss + "  0"; else ss = ss + "  1";
            if (points[0] == 1) ss = ss + "  0 \r\n"; else ss = ss + "  1
\r\n";
            textBox1.AppendText(ss);
        }

        static public byte[] tim = new byte[4] { 1, 6, 11, 16 };
        public Form1()
        {
```

```

InitializeComponent();
for (int j = 0; j < 5; j++) AddPoints(tim);
textBox1.Text = "  T3  T2  T1  Тим \r\n";
}

private void button1_Click(object sender, EventArgs e)
{
    if (timer1.Enabled == false) timer1.Enabled = true;
    else timer1.Enabled = false;
}

private void timer1_Tick(object sender, EventArgs e)
{
    if (tim[3] == 19 && tim[2] == 14 && tim[1] == 9 && tim[0] == 4)
tim[3] = 16;
    else if (tim[3] == 16 && tim[2] == 14 && tim[1] == 9 && tim[0] == 4)
tim[3] = 19;
    if (tim[2] == 14 && tim[1] == 9 && tim[0] == 4) tim[2] = 11;
    else if (tim[2] == 11 && tim[1] == 9 && tim[0] == 4) tim[2] = 14;
    if (tim[1] == 9 && tim[0] == 4) tim[1] = 6;
    else if (tim[1] == 6 && tim[0] == 4) tim[1] = 9;
    if (tim[0] == 4) tim[0] = 1; else tim[0] = 4;
    AddText(tim);
    for (int j = 0; j < 5; j++) AddPoints(tim);
}
}
}

```

Результат работа программы представлен на рисунке 3.8.

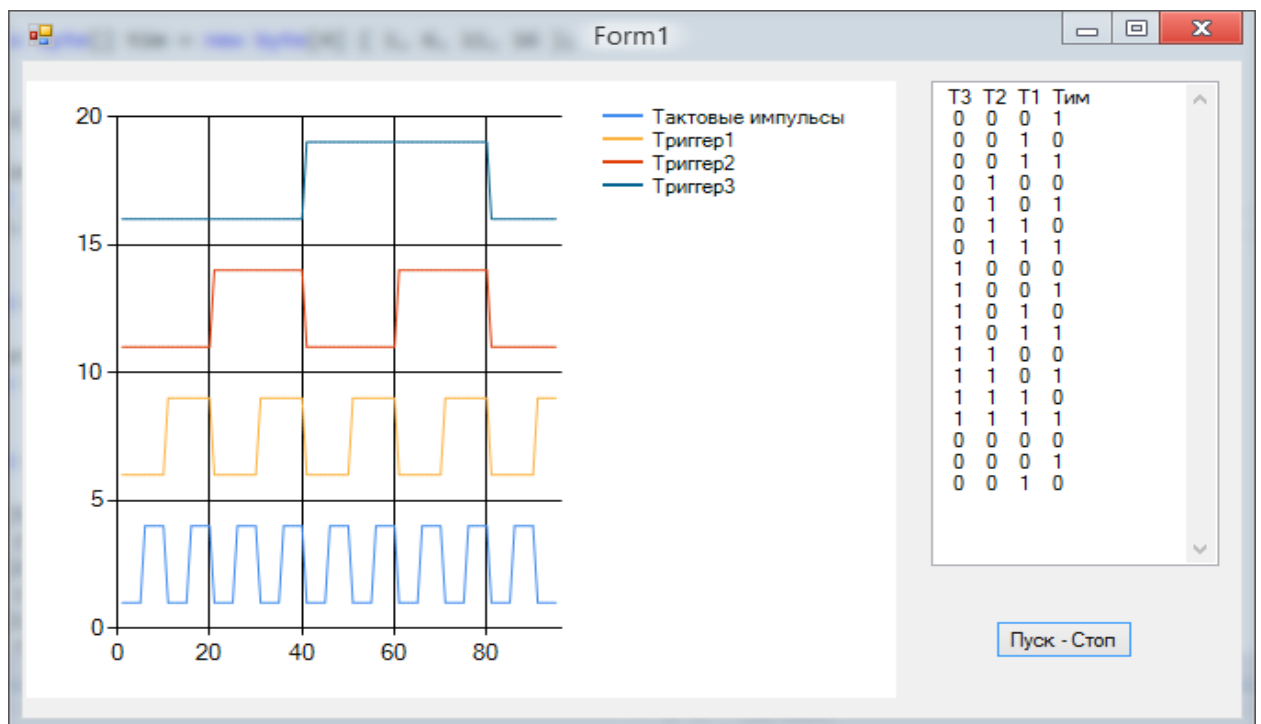


Рисунок 3.8 - Работа программы